

数据结构项目文档

题目：修理牧场

指导教师：张颖

姓名：王亮

学号：1653340

数据结构项目文档

一、题目分析

1. 项目简介
2. 功能需求
3. 设计思路

二、设计

1. 数据结构设计
2. 类的设计
 - 2.1 HuffmanNode
 - 2.2 HuffmanTree
 - 2.3 MinHeap
 - 2.4 LinkedStack
3. 主函数设计

三、实现

1. 构造Huffman树
 - 1.1 流程图
 - 1.2 相关代码
2. 计算Huffman树的非叶结点权值之和
 - 2.1 相关代码

四、测试

1. 功能测试
2. 边界测试
 - 2.1 只有一个木块的情况
3. 出错测试
 - 3.1 木块数量输入错误

一、题目分析

1. 项目简介

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 L_i 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 L_i 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）。

2. 功能需求

1. 输入格式：输入第一行给出正整数N ($N \leq 10^4$)，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。
2. 输出格式：输出一个整数，即将木头锯成N块的最小花费。

3. 设计思路

分析问题，如果能让总花费最少，应该先完成最长的木条的切割任务，其次完成次长木条的切割任务...这个结构与哈夫曼树的思想相同，即权值最大的叶结点位于举例根结点最大的位置。求最小花费实际上就是求哈夫曼树的非叶结点权值之和。

因此问题转化为，用哈夫曼树存储木块长度数据，求此哈夫曼树的非叶结点权值之和。

具体的设计思路为：

首先确定项目采用哈夫曼树作为数据结构，定义类的成员变量和成员函数；然后实现计算哈夫曼树的非叶结点权值之和的函数；最后完成主函数以验证程序的功能并得到运行结果。

二、设计

1. 数据结构设计

如设计思路中所述，求解此问题选择使用**哈夫曼树**结构存储木块长度。

构造哈夫曼树的过程，由于需要依次选择权值最小和次小的结点（非叶结点以其所有子孙叶结点的权值之和作为自身的权值），所以需要**一个最小堆**。

除此之外，需要计算哈夫曼树的非叶结点权值之和，因此需要访问树的所有非叶子结点。我采用了非递归的中序遍历方式，因此需要一个栈存储结点，这里直接调用前面题目中已经实现的**链式栈**。

2. 类的设计

2.1 HuffmanNode

哈夫曼树的结点。

数据域采用模板定义方式，存储结点数据。

指针域包括三个指针，分别指向左孩子、右孩子和父结点。父结点主要在构造哈夫曼树的过程中使用。

```
//Huffman树结点的类定义
template <typename E>struct HuffmanNode{
```

```

    E data;
    HuffmanNode<E> *leftChild, *rightChild, *parent;
    HuffmanNode():leftChild(NULL), rightChild(NULL), parent(NULL){} //构造函数
数
    HuffmanNode(E elem, HuffmanNode<E> * pr, HuffmanNode<E>
*left, HuffmanNode<E> *right)
        :data(elem), parent(pr), leftChild(left), rightChild(right){}
    //重载操作符:
    bool operator > (HuffmanNode<E> right){
        return data>right.data;
    }
    bool operator >= (HuffmanNode<E> right){
        return (data>right.data) || (data==right.data);
    }
    bool operator < (HuffmanNode<E> right){
        return data<right.data;
    }
    bool operator <= (HuffmanNode<E> right){
        return (data<right.data) || (data==right.data);
    }
    bool operator == (HuffmanNode<E> right){
        return data==right.data;
    }
};

```

2.2 HuffmanTree

成员数据包含一个指向根结点的指针。

用到的成员函数主要包括：构造huffman树的构造函数、计算非叶结点权值之和的函数。

```

//Huffman树类定义
template <typename E>class HuffmanTree{
public:
    HuffmanTree(E w[], int n);
    ~HuffmanTree(){
        deleteTree(root);
    }
    HuffmanNode<E>* getRoot(){
        return root;
    }
    void output(HuffmanNode<E> * t, string str, ostream &out);
    E getWPL(); //计算非叶结点权值之和
protected:
    HuffmanNode<E> *root;
    void deleteTree(HuffmanNode<E> * t);
    void mergeTree(HuffmanNode<E> *ht1, HuffmanNode<E> *ht2,
        HuffmanNode<E> *& parent);

```

```
};
```

2.3 MinHeap

最小堆用于构造Huffman树。

```
template < /*class T*/typename Item> class MinHeap{// T为关键码的数据类型, Item
为记录的结构类型
public:
    MinHeap(int sz = DefaultSize);//构造函数: 建立空堆
    MinHeap(Item arr[], int n);      //构造函数: 通过一个数组建堆
    ~MinHeap(){
        delete []heap;
    }
    bool Insert(const Item &x);
    bool RemoveMin(Item &x);
    bool IsEmpty()const{
        return currentSize == 0;
    }
    bool IsFull()const{
        return currentSize == maxHeapSize;
    }
    void MakeEmpty(){
        currentSize = 0;
    }

    void output(){//自定义函数, 顺序输出最小堆元素
        for(int i = 0; i<currentSize; i++)
            cout<<heap[i]<<" ";
        cout<<endl;
    }

private:
    Item *heap;                //存放最小堆中元素的数组
    int currentSize;           //最小堆中当前元素个数
    int maxHeapSize;           //最小堆最多允许元素个数
    void siftDown(int start, int m);//从start到m下滑调整成为最小堆
    void siftUp(int start);      //从start到0上滑调整成为最小堆
};
```

2.4 LinkedStack

链式栈用于实现对Huffman树的非递归中序遍历, 从而计算Huffman树的非叶结点权值之和。

相关类的设计在前面题目中已进行过阐述, 这里不再赘述。

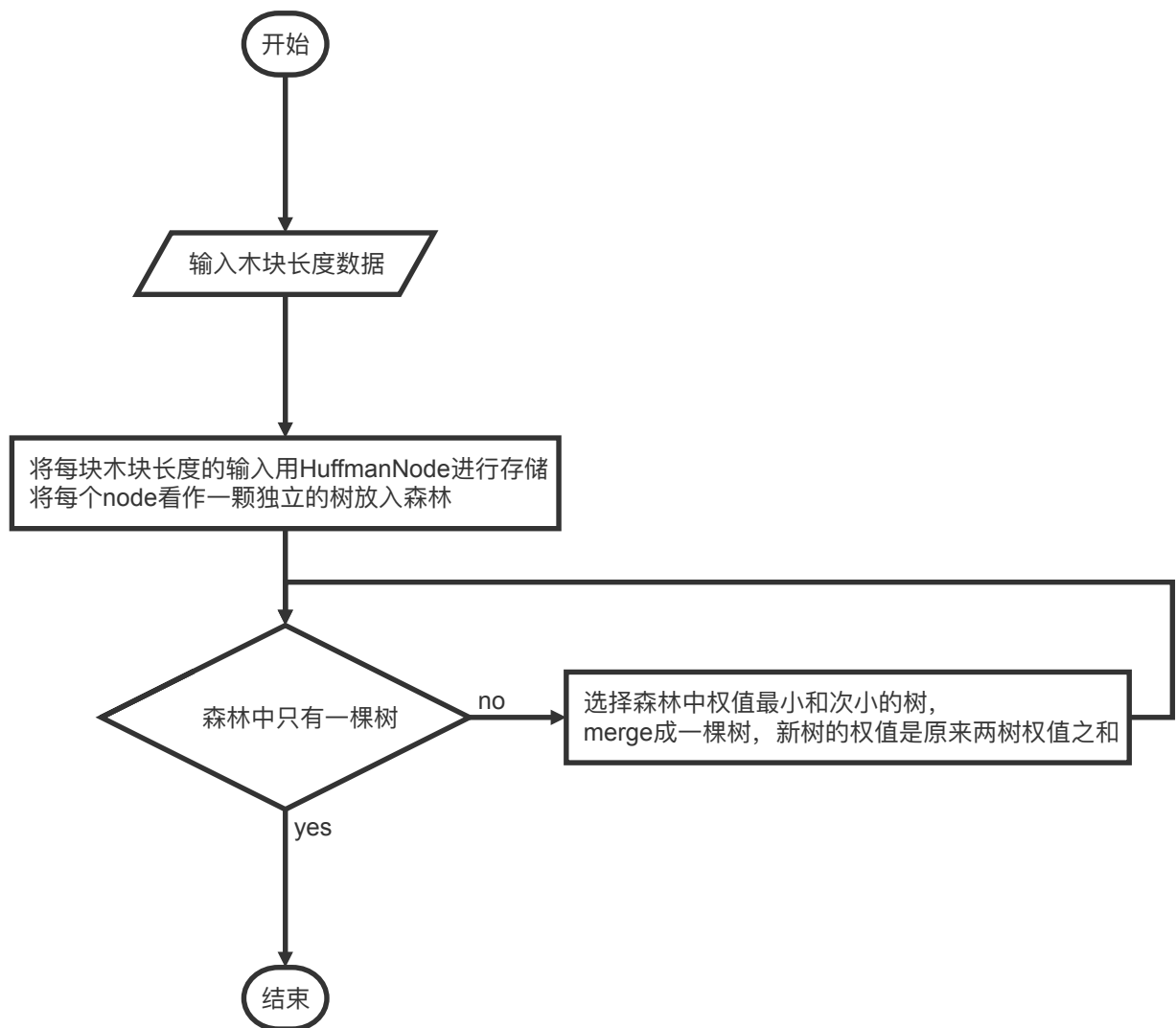
3. 主函数设计

将木块长度信息用Huffman树的数据结构进行存储, 然后计算其非叶结点权值之和并输出即可。

三、实现

1. 构造Huffman树

1.1 流程图



1.2 相关代码

```
template <typename E> HuffmanTree<E>::HuffmanTree(E w[], int n){
    //给出n个权值w[0]~w[n-1], 构造Huffman树
    MinHeap< HuffmanNode<E> > hp;    //使用最小堆存放森林的结点
    HuffmanNode<E> *parent, *first, *second, temp;
    HuffmanNode<E> *NodeList/* = new HuffmanNode<E>[n]*/; //不宜一次性建立森林, 否则析构函数一个一个删结点时操作系统断言出错
```

```

int i;
for (i = 0; i < n; i++){//按棵逐步建立森林中的树木，并作为Huffman树的叶结点。
数据放入森林
    NodeList = new HuffmanNode<E>;
    NodeList->data = w[i];
    NodeList->leftChild = NULL;
    NodeList->rightChild = NULL;
    NodeList->parent = NodeList;//父指针指向自己，信息入堆后，出堆时可以找到对
应结点
    hp.Insert(*NodeList);    //森林信息插入最小堆中
}
for (i = 0; i < n-1; i++){    //n-1趟，建Huffman树
    hp.RemoveMin(temp);        //根权值最小的树
    first=temp.parent;        //first指向对应的最小结点
    hp.RemoveMin(temp);        //根权值次小的树
    second=temp.parent;        //second指向对应的次小结点
    mergeTree(first, second, parent);    //合并
    hp.Insert(*parent);        //新结点插入堆中
}
root = parent;    //建立根结点
//cout<<"    root = "<<root->data<<endl;
//output(root,string(),cout);
}

```

2. 计算Huffman树的非叶结点权值之和

采用非递归的中序遍历方式遍历Huffman树，如果结点不是不是叶结点，则把该结点的值加和。

2.1 相关代码

```

/*
 *@brief: 输出HuffmanTree的非叶结点权值之和
 *计算方法: 非递归中序遍历所有结点，如果不是叶结点，则把结点的值加起来
 */
template <typename E>
E HuffmanTree<E>::getWPL()
{
    E sumWPL = 0;

    HuffmanNode<E>* p = root;
    LinkedStack<HuffmanNode<E>*> S;
    do {
        while (p != NULL) {    //遍历指针向左下移动
            S.Push (p);        //孩子树沿途结点进栈

            p = p->leftChild;
        }
        if (!S.IsEmpty()) {    //栈不空时退栈
            S.Pop (p);    //visit (p);    //退栈，访问

```

```

        if(p->leftChild || p->rightChild)
            sumWPL += p->data;
        p = p->rightChild;    //遍历指针进到右子女
    }
} while (p != NULL || !S.IsEmpty ());

return sumWPL;
}

```

四、测试

1. 功能测试



```

8
4 5 1 2 1 3 1 1
49
Program ended with exit code: 0

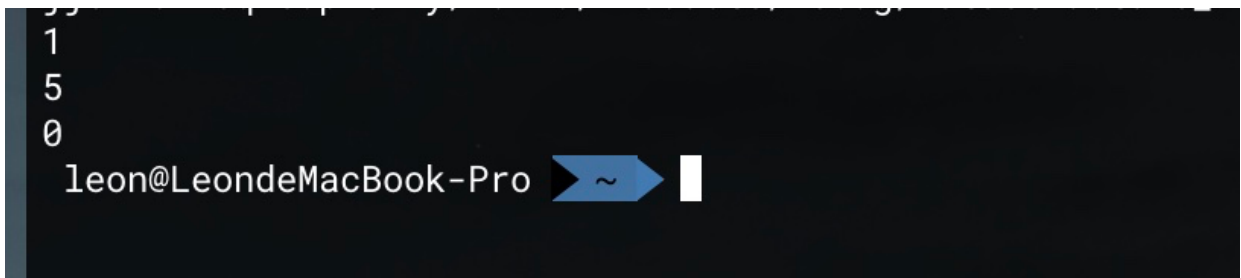
```

测试结果正确。

2. 边界测试

2.1 只有一个木块的情况

测试结果：



```

1
5
0
leon@LeondeMacBook-Pro ~

```

只有一个木块的情况下不需要切割，所以花费为0，测试结果正确。

3. 出错测试

3.1 木块数量输入错误

测试结果：

```
0
木块数量是一个正整数，请重新输入： -1
木块数量是一个正整数，请重新输入： 2
5 6
11
leon@LeondeMacBook-Pro ~
```

当输入的sumBlocks不是一个正整数时，要求用户重新输入，直到符合正整数要求，测试结果正确。