项目说明文档

**数据结构课程设计**

**——修理牧场**

作 者 姓 名： 吴英豪

学 号： 1953608

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

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。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功能分析

(1)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。

(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)项目示例:

8

4 5 1 2 1 3 1 1

49

2.设计

2.1数据结构设计

想要得到最终的n块长度指定的木头可以有多种锯木头的方法，但是不同的锯法所需要的花费是不同的。考虑到最终花费与每次锯出木头的长度有关，中间过程可能不同但最终得到的木块长度都需要和用户指定的值保持一致，所以笔者采用Huffman树和优先级队列来完成此系统。

用户输入的n块木头长度最终将作为Huffman树的叶结点，而其他可能的长度(中间过程值)将作为非叶结点。

构造Huffman树需要将所有木头的长度排序，每次选择长度(权值)最小的木头，所以采用优先级队列来存放这些有序的数。

2.2 类结构设计

考虑到代码的复用性，笔者采用模板类的形式完成系统，模板类能有有效支持不同的数据类型。

系统涉及的类有：优先级队列Queue<T>,哈夫曼树HuffmanTree<T>.由于优先级队列和哈夫曼树可以采用相同类型的结点，所以采用HuffmanNode<T>类同时作为HuffmanTree<T>类和Queue<T>的结点。

由于哈夫曼树HuffmanTree和优先级队列Queue需要访问HuffmanNode中的成员，所以将HuffmanTree和Queue设置为HuffmanNode的友元。

2.3成员与操作设计

(1)结点类(HuffmanNode)

//Huffman结点

template<class T>

class HuffmanNode

{

private:

HuffmanNode<T>\* \_leftChild;

HuffmanNode<T>\* \_rightChild;

//用于队列之中的指针

HuffmanNode<T>\* \_next;

HuffmanNode<T>\* \_front;

T \_data;

public:

HuffmanNode(T data=0);

~HuffmanNode();

bool operator<(HuffmanNode<T>& node);

friend class HuffmanTree<T>;

friend class Queue<T>;

};

(2)霍夫曼树(HuffmanTree)

//Huffman树

template<class T>

class HuffmanTree

{

private:

HuffmanNode<T>\* \_root;

T \_min\_value;

protected:

void deleteNode(HuffmanNode<T>\* node);

public:

HuffmanTree();

HuffmanTree(T data[],int num);

~HuffmanTree();

T getMinValue(HuffmanNode<T>\* node,int height);

HuffmanNode<T>\* getStart() { return \_root->\_next; };

};

(3)优先级队列类（Queue）

//以Huffman结点为元素的双端队列

template<class T>

class Queue

{

private:

HuffmanNode<T>\* \_top; //队首指针

HuffmanNode<T>\* \_last;

protected:

void sort();

public:

Queue();

~Queue();

void push(T data);

bool pop(T& data);

};

2.4文件设计

本系统支持windows系统和Linux系统。

(1)Windows系统

windows系统的相关文件在文件夹07\_1953608\_WuYinghao\_Windows中，

其中类的定义以及其成员函数的定义在07\_1953608\_WuYinghao.h中;main函数以及相关的函数定义在07\_1953608\_WuYinghao.cpp中.07\_1953608\_WuYinghao.exe为Windows平台下的可执行文件。

**07\_1953608\_WuYinghao.cpp和07\_1953608\_WuYinghao.h的编码格式均为**

**简体中文(GB2312)-代码页 936**

(2)Linux 系统

Linux系统的相关文件在文件夹07\_1953608\_WuYinghao\_Linux中,

其中类的定义以及其成员函数的定义在07\_1953608\_WuYinghao\_Linux.h中;main函数以及相关的函数定义在07\_1953608\_WuYinghao\_Linux.cpp中.07\_1953608\_WuYinghao.out为Linux平台下的可执行文件。

**07\_1953608\_WuYinghao\_Linux.cpp和07\_1953608\_WuYinghao\_Linux.h的编码格式均为Unicode(UTF-8 无签名)-代码页65001**

3.实现

3.1算法设计及整体实现

3.1.1 文字说明

现需要按照合理的方法切割木头，使木头的总造价最低。考虑到无论采取什么方法，最终每段木头的长度都应该和用户给定的值保持一致，故采用Huffman树和优先级队列实现此题。

首先需要用户输入相关信息，将这些数值进入优先级队列，并按照从队首到队尾长度依次增加的顺序排序，然后根据优先级队列构造Huffman树，Huffman树的根结点的值即为最终的结果(最小开销)。

3.1.2 核心代码展示

(1)main函数部分

cout << "请输入一个正整数N(1<=N<=10^4),来表示木头的块数" << endl;

int N;

cin >> N;

checkInput(N);

cout << "请输入每一块木头的长度:" << endl;

try

{

int \*data\_array = new int[N];

for (int i = 0; i < N; i++)

{

cin >> data\_array[i];

checkInput(data\_array[i]);

}

HuffmanTree<int>huff(data\_array, N); //构造Huffman树

cout << "最小的花费为: "<<huff.getMinValue(huff.getStart(),0) << endl;

delete[] data\_array;

}

catch (bad\_alloc& ba)

{

cerr << "系统错误!无法生成最优方案" << endl << ba.what() << endl;

system("pause");

exit(1);

}

system("pause");

return 0;

}

**(2)获取最终的最小权值**

template<class T>

T HuffmanTree<T>::getMinValue(HuffmanNode<T>\*node,int height)

{

if (!node) //当前结点为空

{

return 0;

}

else if ((!node->\_leftChild) && (!node->\_rightChild))

{

\_min\_value += node->\_data\*height;

return \_min\_value;

}

else

{

height++;

getMinValue(node->\_leftChild,height);

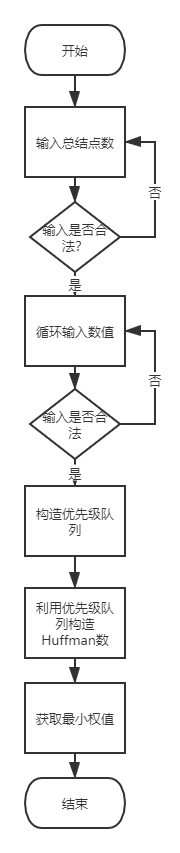
getMinValue(node->\_rightChild,height);

return \_min\_value;

}

}

3.1.3 流程图展示



3.2 优先级队列排序功能

3.2.1 功能说明

队列中每一个结点都拥有一个权值T \_data;，代表一段木头的长度。每当一个队结点从队尾入队时，将此结点的权值与队列中其他元素的权值进行比较，并根据权值对队列中的元素进行重新排序，使最终的结果从队首到队尾构成一个非递减的序列。每当需要有元素出队时，只需要让队首的元素出队，此元素的权值(长度)最小

3.2.2 核心代码展示

template<class T>

void Queue<T>::sort()

{

HuffmanNode<T>\* current = \_top->\_next;

HuffmanNode<T>\* target = \_last->\_next;

T current\_data = current->\_data;

T target\_data = target->\_data;

while (\*current\_data<\*target\_data)

{

current = current->\_next;

current\_data = current->\_data;

}

if (current!= \_last->\_next) //并不是最后一个位置，可以移动

{

HuffmanNode<T>\* temp = \_last->\_next->\_front; //暂存最后的队尾结点

current->\_front->\_next = \_last->\_next;

\_last->\_next->\_front = current->\_front;

current->\_front = \_last->\_next;

\_last->\_next->\_next = current;

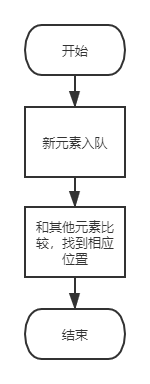
temp->\_next = NULL;

\_last->\_next = temp;

}

}

3.2.3 流程图展示



3.3 构造Huffman树

3.3.1 功能说明

首先将优先级队列都作为一个单独的树，然后选取优先级队列中权值最小的两个结点(队中前两个元素)，将这两个数进行合并，其父节点的权值为两个子女的权值之和，再将父节点入队，优先级队列重新排序。

反复重复上述操作，直到优先级队列中只剩下一个元素，这个对应的结点即为Huffman树的根结点，构造Huffman树成功。最终的最小权值(总花费)即为根结点的参数

3.3.2 核心代码展示

template<class T>

HuffmanTree<T>::HuffmanTree(T data[], int num)

{

\_min\_value = 0;

\_root = NULL;

\_root = new HuffmanNode<T>;

Queue<HuffmanNode<T>\*> queue;

HuffmanNode<T>\* first, \*second;

for (int i = 0; i < num; i++)

{

try

{

HuffmanNode<T>\* newNode = new HuffmanNode<T>(data[i]);

queue.push(newNode);

}

catch (bad\_alloc& ba)

{

cerr << "无法创建树结点" << endl << ba.what() << endl;

}

}

if (1 == num)

{

try

{

queue.pop(first);

HuffmanNode<T>\* parent = new HuffmanNode<T>;

parent->\_data = first->\_data;

parent->\_leftChild = first;

parent->\_rightChild = NULL;

\_root->\_next= parent;

}

catch (bad\_alloc& ba)

{

cerr << "无法创建树节点" << endl << ba.what() << endl;

}

}

for (int j = 0; j < num - 1; j++)

{

queue.pop(first);

queue.pop(second);

try

{

HuffmanNode<T>\* parent = new HuffmanNode<T>;

parent->\_leftChild = first;

parent->\_rightChild = second;

parent->\_data = first->\_data + second->\_data;;

queue.push(parent);

\_root->\_next=parent;

}

catch (bad\_alloc& ba)

{

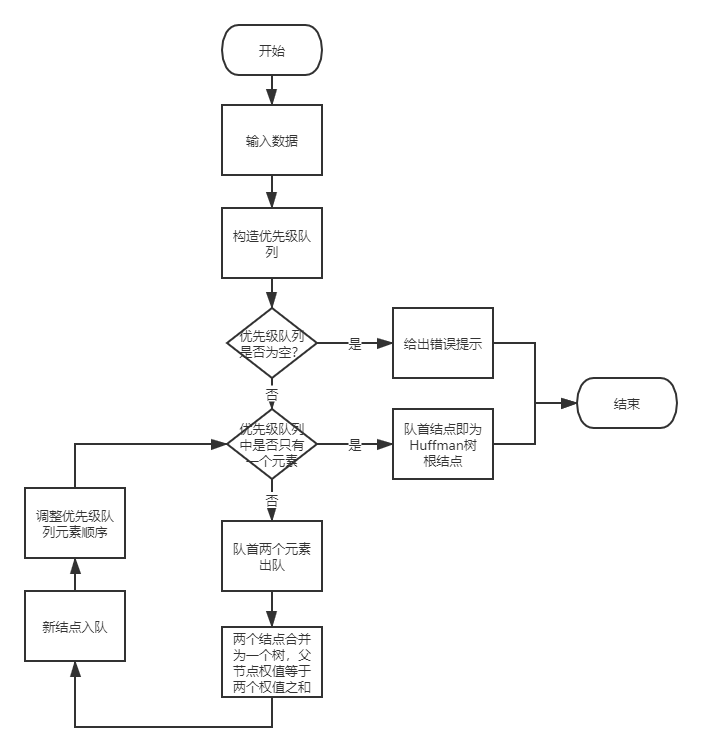
cerr << "无法创建树节点" << endl << ba.what() << endl;

}

}

}

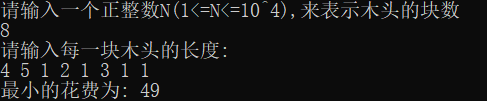
3.3.3 流程图展示



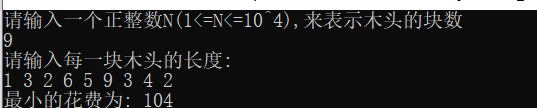
**4.测试**

4.1 功能测试

4.1.1 样例测试



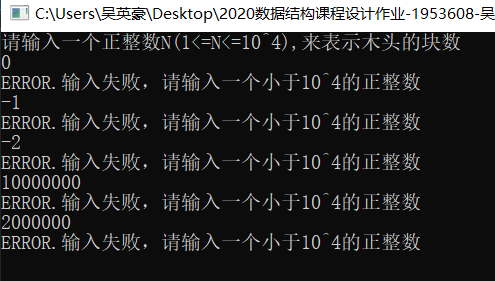
4.1.2 其他相关测试



4.2 边界测试

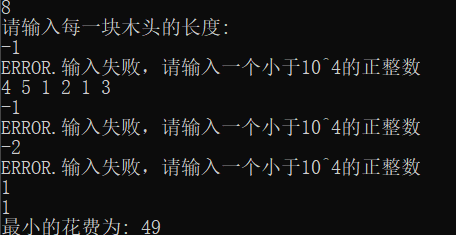
4.2.1 检测木头的块数输入是否合法

题目中要求第一行的数据(木头的块数)为小于10^4的正整数，所以需要进行边界测试。对于不合法的输入值给出相应的提升



**4.2.2 检测每一块木头的长度是否合法**

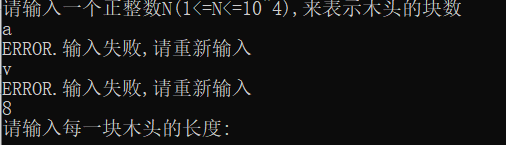
每一块木头的长度输入值必须是正整数，若输入非正整数，需要给出相应的错误提示。



4.3 错误测试

4.3.1 检测木头块数输入是否合法

题目要求输入正整数，若输入字符等非int型的数据，系统能够给出相应的出错提示。



4.3.2 检测每块木头的长度输入是否合法

每一块木头的长度应该为正整数，若输入字符等非int类型的数据，系统能够给出相应的出错提示。

