项目说明文档

数据结构课程设计

——修理牧场

作 者 姓 名： 祝新元

学 号： 1751629

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

目 录

[1 分析 1](#_Toc532324221)

[1.1 项目简介 1](#_Toc532324222)

[1.2 功能分析 1](#_Toc532324223)

[2 设计 2](#_Toc532324224)

[2.1 数据结构设计 2](#_Toc532324225)

[2.2 类结构设计 2](#_Toc532324226)

[2.3 成员与操作设计 2](#_Toc532324227)

[2.4 系统设计 4](#_Toc532324228)

[3 实现 4](#_Toc532324229)

[3.1 构造霍夫曼树的实现 4](#_Toc532324230)

[3.1.1核心代码 4](#_Toc532324231)

[3.2 合并功能的实现 5](#_Toc532324232)

[3.2.1 合并功能核心代码 5](#_Toc532324233)

[3.3 最小堆的结构说明 5](#_Toc532324234)

[3.3.1从上到下调整功能的代码 5](#_Toc532324235)

[3.3.2 插入功能核心代码 6](#_Toc532324236)

[3.3.3 通过数组的构造函数 6](#_Toc532324237)

[3.4 总体的实现 7](#_Toc532324238)

[3.4.1 总体核心代码 7](#_Toc532324239)

[3.4.2 总体系统截屏示例 8](#_Toc532324240)

# 1 分析

## 1.1 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*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。

# 2 设计

## 2.1 数据结构设计

如上所述， 为了解决该问题，采用霍夫曼树的结构来求解每块的木头长度。

## 2.2 类结构设计

霍夫曼树一般包括两个抽象数据类型（ADT）——结点类（HuffmanNode）与霍夫曼树类（HuffmanTree），而两个类之间的耦合关系可以采用嵌套、继承等多种关系。为方便处理，本系统采用struct描述结点类（HuffmanNode），这样使得霍夫曼树类（HuffmanTree）可以访问链表结点。

同时需要最小堆（MinHeap）的来存放森林，因为要用通过它来选择根节点权值最小和次小的两棵树。

## 2.3 成员与操作设计

**链表结点类（**HuffmanNode**）**

**公有成员：**

float data; //数据值

HuffmanNode \*leftChild,\*rightChild,\*parent;//左右子女和父结点指针

**公有操作：**

HuffmanNode():leftChild(NULL),rightChild(NULL),parent(NULL){} //构造函数

HuffmanNode(float elem,HuffmanNode\* left=NULL,

HuffmanNode\* right=NULL,HuffmanNode\* pr=NULL)

:data(elem),leftChild(left),rightChild(right),parent(pr){} //构造函数

**霍夫曼树类**（HuffmanTree）

**私有成员：**

float sum; //将木头锯成N块的最小花费

HuffmanNode \*root; //根结点

**公有操作：**

void mergeTree(HuffmanNode& ht1,HuffmanNode& ht2,HuffmanNode\* &parent);

//合并二叉树

HuffmanTree::HuffmanTree(float w[],int n)

//给出n个权值来构造huffman树

float HuffmanTree::Show();

//输出霍夫曼树

**最小堆（MinHeap）**

**私有成员：**

E \*heap;//存放最小堆中元素的数组

int currentSize;

//最小堆中当前元素的个数

int maxHeapSize;

//最小堆中最多允许元素的个数

**私有操作：**

void siftDown(int start,int m);

//从start到m下滑调整为最小堆

void siftUp(int start);

//从start到0调整成为最小堆

**公有操作：**

MinHeap<E>::MinHeap(int sz=DefaultSize);

//构造函数

MinHeap<E>::MinHeap(E arr[],int n);

//通过一个数组建立堆

MinHeap<E>::~MinHeap()

//析构函数

bool MinHeap<E>::Insert(const E& x);

//将x插入到最小堆中

bool MinHeap<E>::RemoveMin(E& x);

//删除堆顶的最小元素

bool MinHeap<E>::IsEmpty()const

//判断是否是空

bool MinHeap<E>::IsFull()const

//判断是否是满

void MinHeap<E>::MakeEmpty()

//置空堆

void MinHeap<E>::Show();

//输出最小堆中所有值

## 2.4 系统设计

首先输入n块木头的长度，将他们输入到一个数组a里面。通过a数组来构建霍夫曼树得到答案。

# 3 实现

## 3.1 构造霍夫曼树的实现

### 3.1.1核心代码

HuffmanTree::HuffmanTree(float w[],int n) //给出n个权值来构造huffman树

{

//使用最小堆来存放

MinHeap<HuffmanNode> hp;

HuffmanNode \*parent;

HuffmanNode first,second,work;

//初始化总花费sum

sum=0;

//将各树分别插入到最小堆中

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

{

work.data=w[i];

work.leftChild=NULL;

work.rightChild=NULL;

work.parent=NULL;

hp.Insert(work);

}

for(int i=0;i<n-1;i++)

{

hp.RemoveMin(first);//选择根的权值最小的树

hp.RemoveMin(second);//选择根的权值次小的树

mergeTree(first,second,parent);//将他们合并

hp.Insert(\*parent);//重新插入到最小堆中

}

root=parent;//建立根节点

}

## 3.2 合并功能的实现

### 3.2.1 合并功能核心代码

void HuffmanTree::mergeTree(HuffmanNode& ht1,HuffmanNode& ht2,HuffmanNode\* &pr)

{

//合并ht1和ht2，即pr成为ht1和ht2的父结点

pr=new HuffmanNode;

pr->leftChild=&ht1;

pr->rightChild=&ht2;

pr->data=ht1.data+ht2.data;

sum+=pr->data;//刷新总花费

ht1.parent=ht2.parent=pr;

}

## 3.3 最小堆的结构说明

### 3.3.1从上到下调整功能的代码

template<class E>

void MinHeap<E>::siftDown(int start,int m)//从节点start到m从上到下比较，如果子女的值小于

{ //父节点的值则调整

int i=start;

int j=2\*i+1;

E temp=heap[i];

while(j<=m)

{

if(j<m&&heap[j]>heap[j+1])j++; //j->min(heap[j],heap[j+1])

if(temp<=heap[j])break;

else

{

heap[i]=heap[j];

i=j;

j=2\*j+1;

}

}

heap[i]=temp;

}

### 3.3.2 插入功能核心代码

template<class E>

bool MinHeap<E>::Insert(const E& x)

{

if(currentSize==maxHeapSize)

{

cerr<<"Heap Full!"<<endl;

return false;

}

heap[currentSize]=x;

siftUp(currentSize);

currentSize++;

return true;

}

### 3.3.3 通过数组的构造函数

template<class E>

MinHeap<E>::MinHeap(E arr[],int n)

{

maxHeapSize=(DefaultSize<n)?n:DefaultSize;

heap=new E[maxHeapSize];

if(heap==NULL)

{

cerr<<"堆储存分配失败！"<<endl;

exit(1);

}

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

{

heap[i]=arr[i];

}

currentSize=n;

int currentPos=(currentSize-2)/2;

while(currentPos>=0)

{

siftDown(currentSize,currentSize-1);

currentSize--;

}

}

## 3.4 总体的实现

### 3.4.1 总体核心代码

int n;

cin>>n;

float a[10000];

//输入每块木头的长度

//输入

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

{

cin>>a[i];

}

HuffmanTree ht(a,n);

//得到最小花费

cout<<ht.Show()<<endl;

### 3.4.2 总体系统截屏示例

