项目说明文档

数据结构课程设计

——修理牧场

作 者 姓 名： 杨煜

学 号： 1850217

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](#_Toc26872889)

[1.1 项目简介 1](#_Toc26872890)

[1.2 项目要求 1](#_Toc26872891)

[2 设计 1](#_Toc26872892)

[2.1 数据结构设计 1](#_Toc26872893)

[2.2 最小堆结构设计 1](#_Toc26872894)

[2.3 成员与操作设计 2](#_Toc26872895)

[3 实现 2](#_Toc26872896)

[3.1 构造函数的实现 2](#_Toc26872897)

[3.1.1 构造函数 2](#_Toc26872898)

[3.1.2 构造函数核心代码 3](#_Toc26872899)

[3.2 下滑调整函数的实现 3](#_Toc26872900)

[3.2.1 下滑调整函数 3](#_Toc26872901)

[3.2.2 下滑调整函数核心代码 3](#_Toc26872902)

[3.3 向上调整的实现 4](#_Toc26872903)

[3.3.1 向上调整 4](#_Toc26872904)

[3.3.2 向上调整核心代码 4](#_Toc26872905)

[3.4 插入的实现 5](#_Toc26872906)

[3.4.1 插入 5](#_Toc26872907)

[3.4.2 获取头节点核心代码 5](#_Toc26872908)

[3.5 移除函数的实现 5](#_Toc26872909)

[3.5.1 移除函数 5](#_Toc26872910)

[3.5.2 交集函数核心代码 5](#_Toc26872911)

[3.6 总体流程的实现 6](#_Toc26872912)

[3.6.1 总体流程 6](#_Toc26872913)

[3.6.2 总体流程核心代码 6](#_Toc26872914)

[3.6.3 总体流程图 8](#_Toc26872915)

[4 测试 8](#_Toc26872916)

[4.1 功能测试 8](#_Toc26872917)

[4.1.1 一般情况 8](#_Toc26872918)

[4.1.2 零个输入 9](#_Toc26872919)

[4.1.3 一个输入 10](#_Toc26872920)

[4.2 出错测试 11](#_Toc26872921)

[4.2.1 含有负整数 11](#_Toc26872922)

# 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）。

## 项目要求

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

如上述项目简介所述，为了达成所锯的木头的最小，是Huffman树的应用，为了达到金额最小则需要使用最小堆的数据结构。主体的部分并不复杂，只需在优先队列中连续弹出两个内容，相加之后再插入，依次往复。

## 2.2 最小堆结构设计

在实现最小堆时，将插入元素进行相应的调整是极其重要的。堆是一种完全二叉树的数据结构，故而它的存储结构便可以通过使用数组来存储数据。对它进行插入操作的时间复杂度为O（log n），效率高。因此，可以使用最小堆来完成优先队列的建立。

## 2.3 成员与操作设计

template<class E>

class Heap

{

public:

Heap(int sz = 100000);//构造函数

~Heap() { delete[]heap; }//析构函数

bool Insert(const E& x);//插入

E Remove();//移除节点

bool IsEmpty()const//判空

{

return(currentSize == 0) ? true : false;

}

bool IsFull()const//判满

{

return(currentSize == maxSize) ? true : false;

}

int getSize()const { return currentSize; }//返回堆大小

private:

E\* heap;//指向堆的指针

int currentSize;//当前大小

int maxSize;//最大大小

void siftDown(int start, int m);//向下滑动调整

void siftUp(int start);//向上滑动调整。

};

# 3 实现

## 3.1 构造函数的实现

### 3.1.1 构造函数

构造函数构造函数接受一个默认的参数100000，开辟数组空间，当前的堆的大小置为0，构造函数依此完成。

### 3.1.2 构造函数核心代码

template<class E>

Heap<E>::Heap(int sz)

{

maxSize = (100000 < sz) ? sz : 100000;

heap = new E[maxSize];

currentSize = 0;

}

## 3.2 下滑调整函数的实现

### 3.2.1 下滑调整函数

下滑调整函数接受调整位置和终点的参数。置j为i的子节点如果j小于m，同时j大于j+1则调整至其兄弟节点。将i的内容向下调整，直到到达m或者i所在内容小于j所在的内容。

### 3.2.2 下滑调整函数核心代码

template<class E>

void Heap<E>::siftDown(int start, int m)

{

int i = start, j = 2 \* i + 1;

E temp = heap[i];

while (j <= m)

{

if (j<m && heap[j]>heap[j + 1])//比较两个子节点的大小

{

j++;

}

if (temp <= heap[j])//双亲节点小于子节点

break;

else//否则进行调整

{

heap[i] = heap[j];

i = j;

j = 2 \* j + 1;

}

}

heap[i] = temp;

}

## 3.3 向上调整的实现

### 3.3.1 向上调整

向上调整函数接受调整位置参数。置j为i的子节点如果j大于0，则进行调整，将i中的内容向上调整，直到到达根节点或者i所在内容小于j所在的内容

### 3.3.2 向上调整核心代码

template<class E>

void Heap<E>::siftUp(int start)

{

int j = start, i = (j - 1) / 2;

E temp = heap[j];

while (j > 0)

{

if (heap[i] <= temp)

break;

else

{

heap[j] = heap[i];

j = i;

i = (i - 1) / 2;

}

}

heap[j] = temp;

}

## 3.4 插入的实现

### 3.4.1 插入

插入函数接受的参数为所要插入的内容，在当前的下一个位置处插入，最后从这个位置开始向上调整。

### 3.4.2 获取头节点核心代码

template<class E>

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

{

heap[currentSize] = x;

siftUp(currentSize);

currentSize++;//大小加一

return true;

}

## 3.5 移除函数的实现

### 3.5.1 移除函数

将堆栈的根节点输出并使用最后一个子节点来代替。同时进行从0开始到最后位置的向下调整。

### 3.5.2 交集函数核心代码

template<class E>

E Heap<E>::Remove()

{

E x = heap[0];

heap[0] = heap[--currentSize];

siftDown(0, currentSize - 1);

return x;

}

## 3.6 总体流程的实现

### 3.6.1 总体流程

首先申明一个最小堆，将读入的数字存放入堆中，从堆中读出两个数将其相加之后在放入，最后便可以得出最小值。

### 3.6.2 总体流程核心代码

int main()

{

Heap<int> pque;

int n, k;

int flag = 0;

int ans = 0;

cin >> n;

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

{

cin >> k;

if (k < 0)

{

flag = 1;

}

pque.Insert(k);

}

if (flag == 0)

{

if (n == 1)

{

cout << pque.Remove();

}

else

{

while (pque.getSize() > 1)

{

int a, b;

a = pque.Remove();

b = pque.Remove();

ans += a + b;

pque.Insert(a + b);

}

cout << ans;

}

}

else

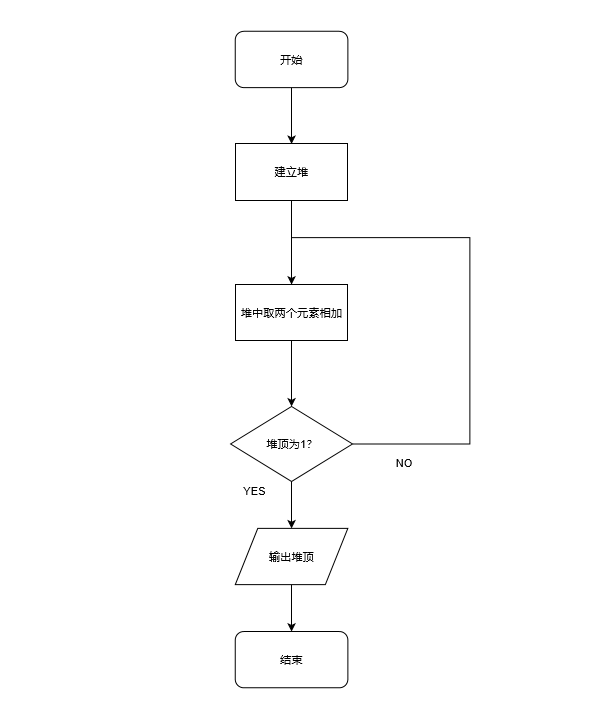
{

cout << "错误，长度不能为负";

}

}

### 3.6.3 总体流程图



# 4 测试

## 4.1 功能测试

### 4.1.1 一般情况

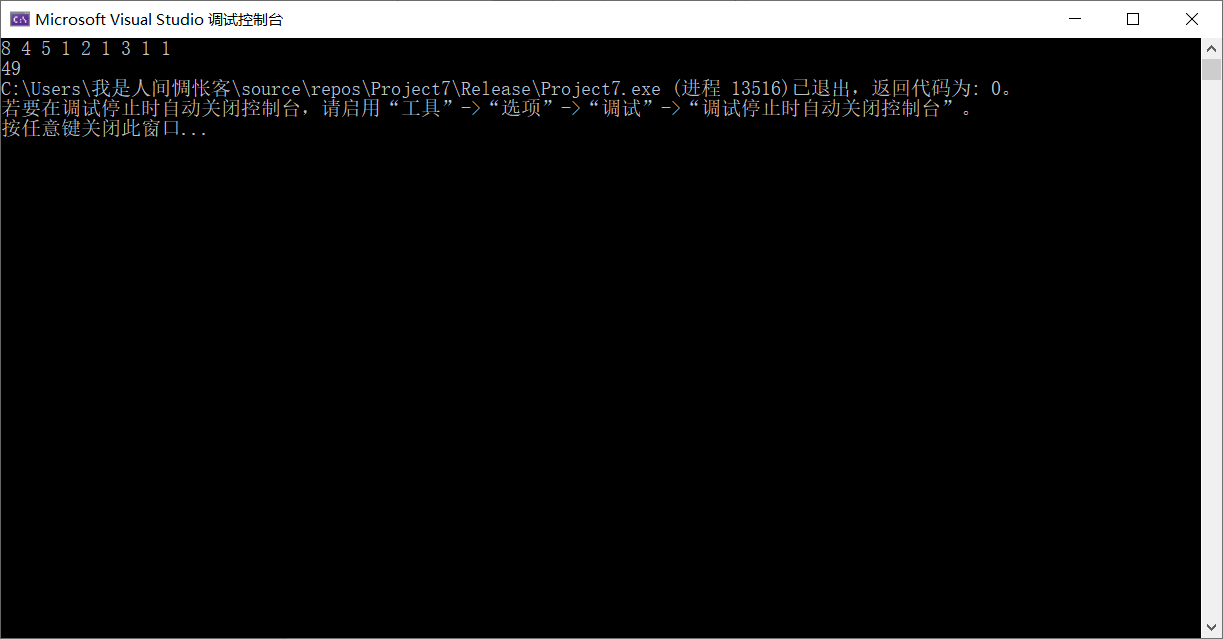
**测试用例**：

8 4 5 1 2 1 3 1 1

**预期结果**：

49

**实验结果**



### 4.1.2 零个输入

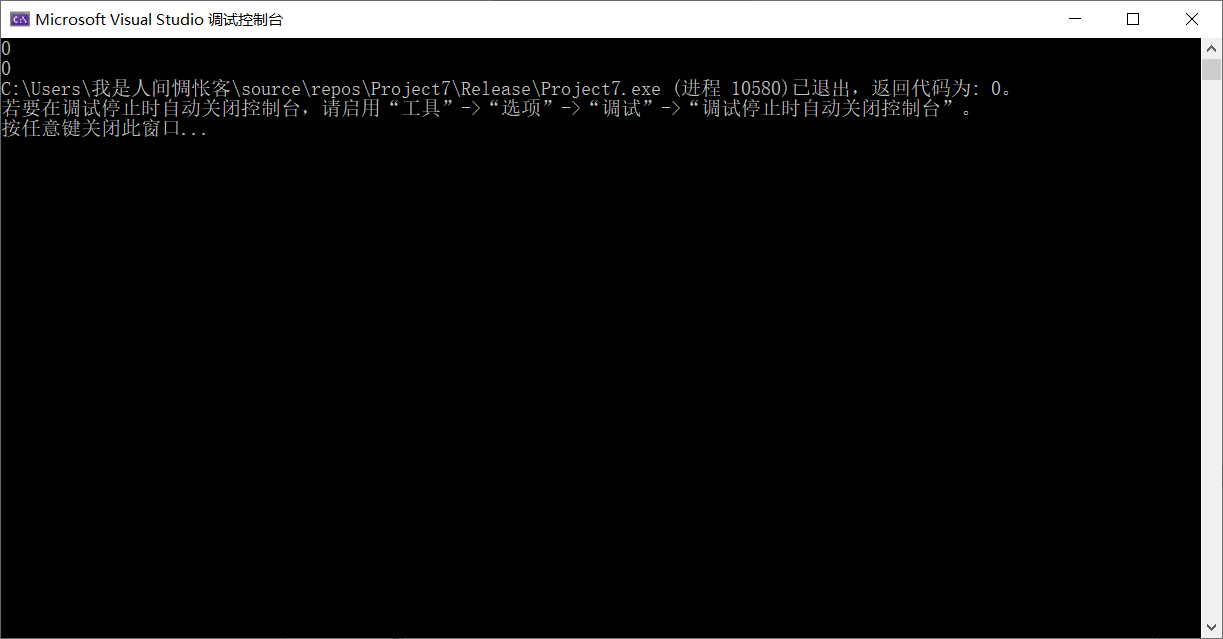
**测试用例：**

0

**预期结果：**

**0**

**实验结果：**



### 4.1.3 一个输入

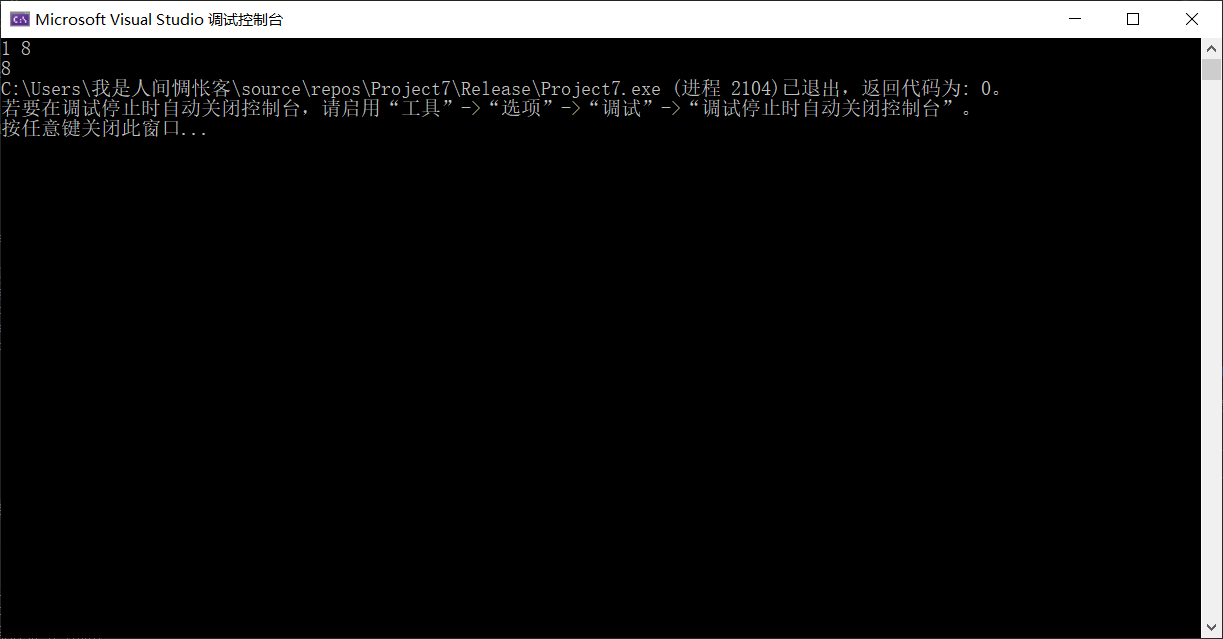
**测试用例：**

1 8

**预期结果：**

8

**实验结果：**



## 4.2 出错测试

### 4.2.1 含有负整数

**测试用例：**

3 -2 5 -3

**预期结果：**程序给出提示信息，程序正常运行不崩溃。

**实验结果：**

