项目说明文档

数据结构课程设计

——修理牧场

同济大学

Tongji University

目 录

[1 分析 1](#_Toc4596)

[1.1 背景分析 1](#_Toc22175)

[1.2 功能分析 1](#_Toc26600)

[2 设计 1](#_Toc30623)

[2.1 数据结构设计 1](#_Toc6514)

[2.2 类结构设计 1](#_Toc11781)

[2.3 成员与操作设计 1](#_Toc31574)

[2.4 系统设计 2](#_Toc3415)

[3 实现 2](#_Toc14119)

[3.1 排序功能的实现 2](#_Toc6333)

[3.1.1 排序功能流程图 2](#_Toc3671)

[3.1.2 排序功能核心代码 3](#_Toc30761)

[3.2 交换功能的实现 5](#_Toc31480)

[3.2.1 交换功能流程图 5](#_Toc29637)

[3.2.2 交换功能核心代码 5](#_Toc2470)

[3.3 合并功能的实现 6](#_Toc3492)

[3.3.1 合并功能流程图 6](#_Toc23292)

[3.3.2 合并功能核心代码 6](#_Toc16045)

[3.7总体系统实现 7](#_Toc32047)

[3.7.1 总体系统流程图 7](#_Toc1661)

[3.7.2总体系统核心代码 8](#_Toc23298)

[4 测试 11](#_Toc4115)

[4.1 功能测试 11](#_Toc3377)

[4.2 出错测试 11](#_Toc25681)

[4.2.1 输入的段数小于等于1 11](#_Toc18742)

[4.2.2 输入的数字小于1 11](#_Toc6830)

# 1 分析

## 1.1 背景分析

Huffman树是一种求最小代价的算法，给定N个权值作为N个[叶子结点](https://baike.baidu.com/item/%E5%8F%B6%E5%AD%90%E7%BB%93%E7%82%B9/3620239)，构造一棵二叉树，若该树的带权路径长度达到最小，称这样的二叉树为最优二叉树，也称为哈夫曼树，哈夫曼树是带权路径长度最短的树，权值较大的结点离根较近。

## 1.2 功能分析

要建立一个Huffman树，首先应该将各个数据按升序进行排序，放入数组中。然后放入对应的“根-叶”单支树中，用一个链表将这些树按升序穿起来，设计一个交换的类函数，一个合并的类函数，不断合并头端的两个树，比较代价交换到对应的位置。在重复上述的过程。综上所述，我们有一个排序功能，一个合并功能，一个交换功能

# 

# 2 设计

## 2.1 数据结构设计

如上功能分析所述，该系统中的树除了叶节点外都是满二叉树。同时，为了实现排序简易，在链表的第一个结点之前附加一个头结点，这样就使得增加或者删除头结点与处理其他结点方法相同，使得程序简洁。

## 2.2 类结构设计

经典的链表一般包括两个抽象数据类型（ADT）——链表结点类（date）与链表类（list），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描述链表结点类(date)，这样使得链表结点类（list）可以访问链表结点。

## 2.3 成员与操作设计

class huffman

{

public:

huffman\* left, \*right; //左右子女

huffman\* link;

int total; //该节点的代价

huffman() { total = 0; left == NULL; right == NULL; link = NULL; } //构造函数1

huffman(int a) { left = new huffman; left->total = a; total = a; right = NULL; }//构造函数2

void add() { total = left->total + right->total; } //计算该节点的代价

void swap(huffman \* &a, huffman \* &b); //交换该节点和之后的节点

int combine(int &number,huffman \*&a); //将节点合并，自己和后面一个节点

int addtotal(int &a); //计算以自己为根节点的Huffman代价

## };

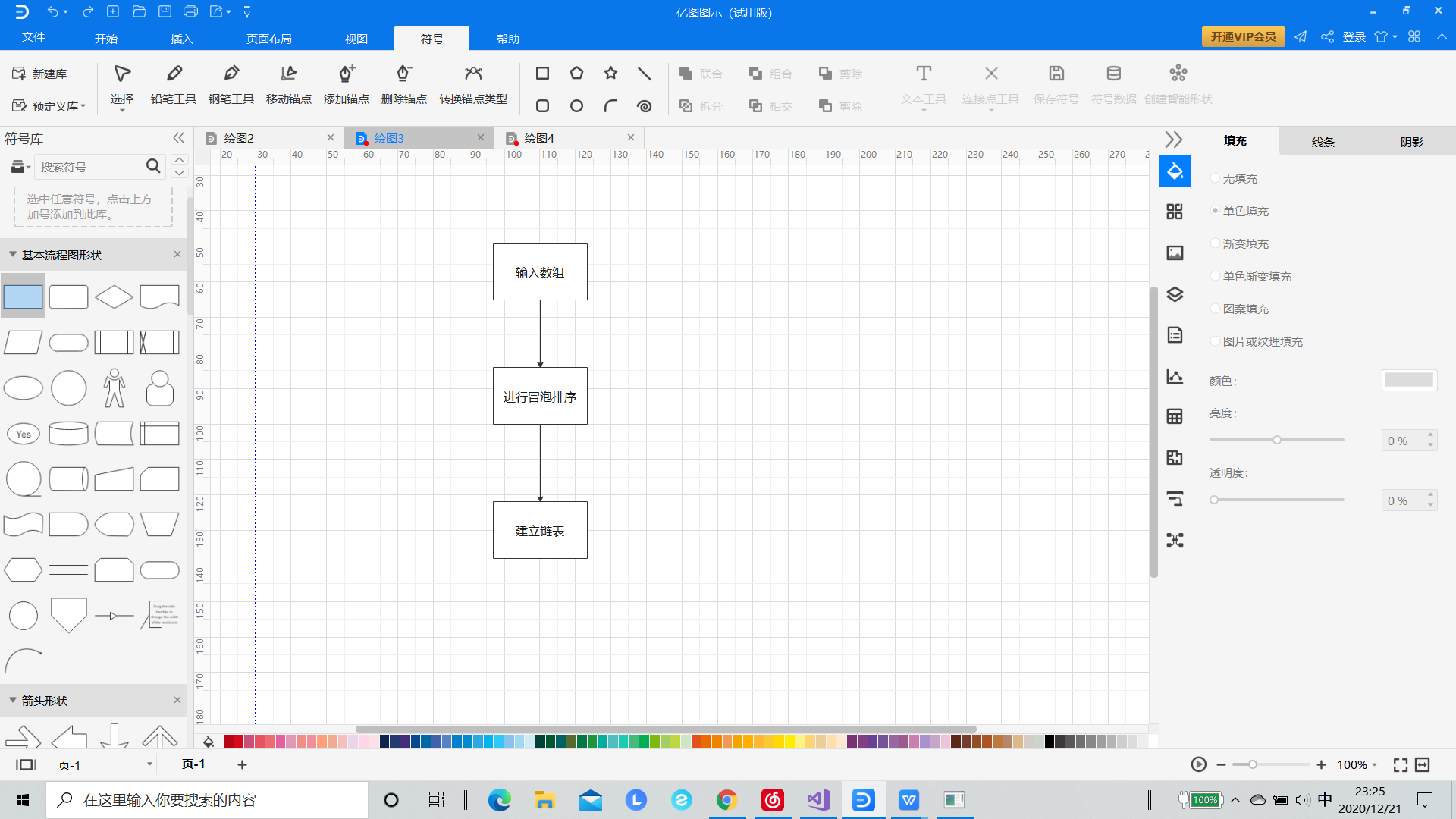
## 2.4 系统设计

系统首先调用排序功能将长度按升序的序列排出，然后赋值给对应的树，再进行Huffman树的生成

# 3 实现

## 3.1 排序功能的实现

### 3.1.1 排序功能流程图



### 3.1.2 排序功能核心代码

采用冒泡法进行排序

int putnumber[1024];

int number = 0;

//进行输入和排序，从小到大

cin >> number;

if (number <= 1) { cout << "输入的数小于或等于1 错误" << endl; }//边界条件

else

{

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

{

cin >> putnumber[n];

}

if (number == 2)

{

cout << putnumber[0] + putnumber[1] << endl;

}

else

{

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

{

int temp;

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

{

if (putnumber[m] > putnumber[m + 1])

{

temp = putnumber[m];

putnumber[m] = putnumber[m + 1];

putnumber[m + 1] = temp;

}

}

}

//通过从小到大的数组建立一个从小到大的链表来表示森林

huffman \*head, \*current;

head = new huffman;

current = new huffman(putnumber[0]);

head->link = current;

for (int n = 1; n < number; n++)

{

current->link = new huffman(putnumber[n]);

current = current->link;

### }

## 3.2 交换功能的实现

### 3.2.1 交换功能流程图

### 

### 3.2.2 交换功能核心代码

void huffman::swap(huffman \* &a,huffman \* &b)//返回两个指针方便再次链接

{

if (link == NULL) { cout<<"link=null"<<endl; }

if (link->link != NULL) //后面不为空节点

{

huffman\* p1;

p1 = link;

link = link->link;

p1->link = this;

a = link;

b = p1;

}

else //后面为空节点

{

huffman\* p1;

p1 = link;

link = NULL;

p1->link = this;

a = link;

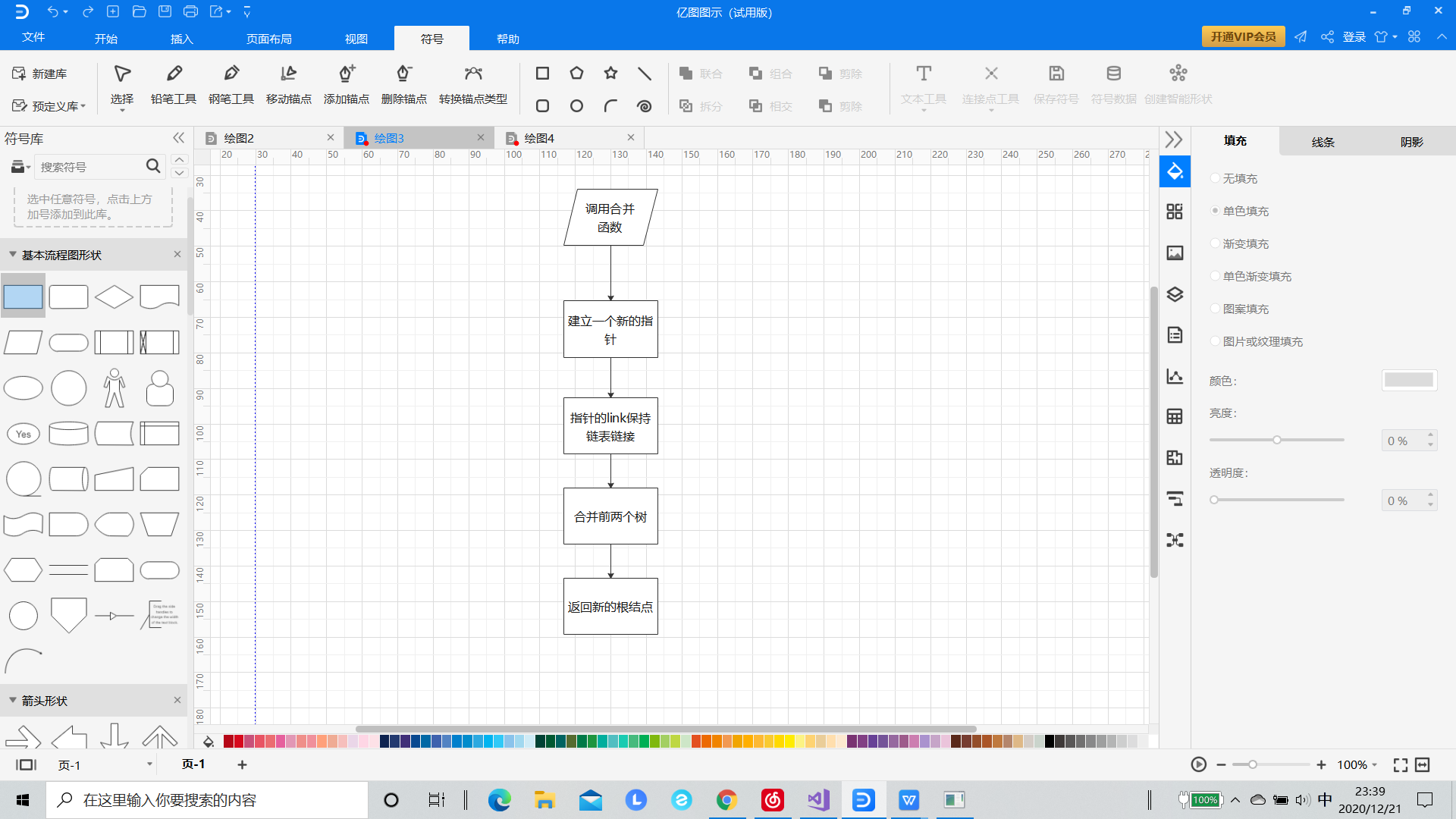
b = p1;

}

}

## 3.3 合并功能的实现

### 3.3.1 合并功能流程图



### 3.3.2 合并功能核心代码

int huffman::combine(int &number,huffman\* &a)

{

if (number < 1) { cout << "number < 1" << endl; return 0; }

huffman \*p1; //创造一个新指针进行相加，并将新指针通过参数返回

huffman \*p2;

huffman \* newone = new huffman;

p1 = this;

p2 = link;

newone->link = link->link;

link->link = NULL;

link = NULL;

newone->left = p1;

newone->right = p2;

newone->add();

number--;

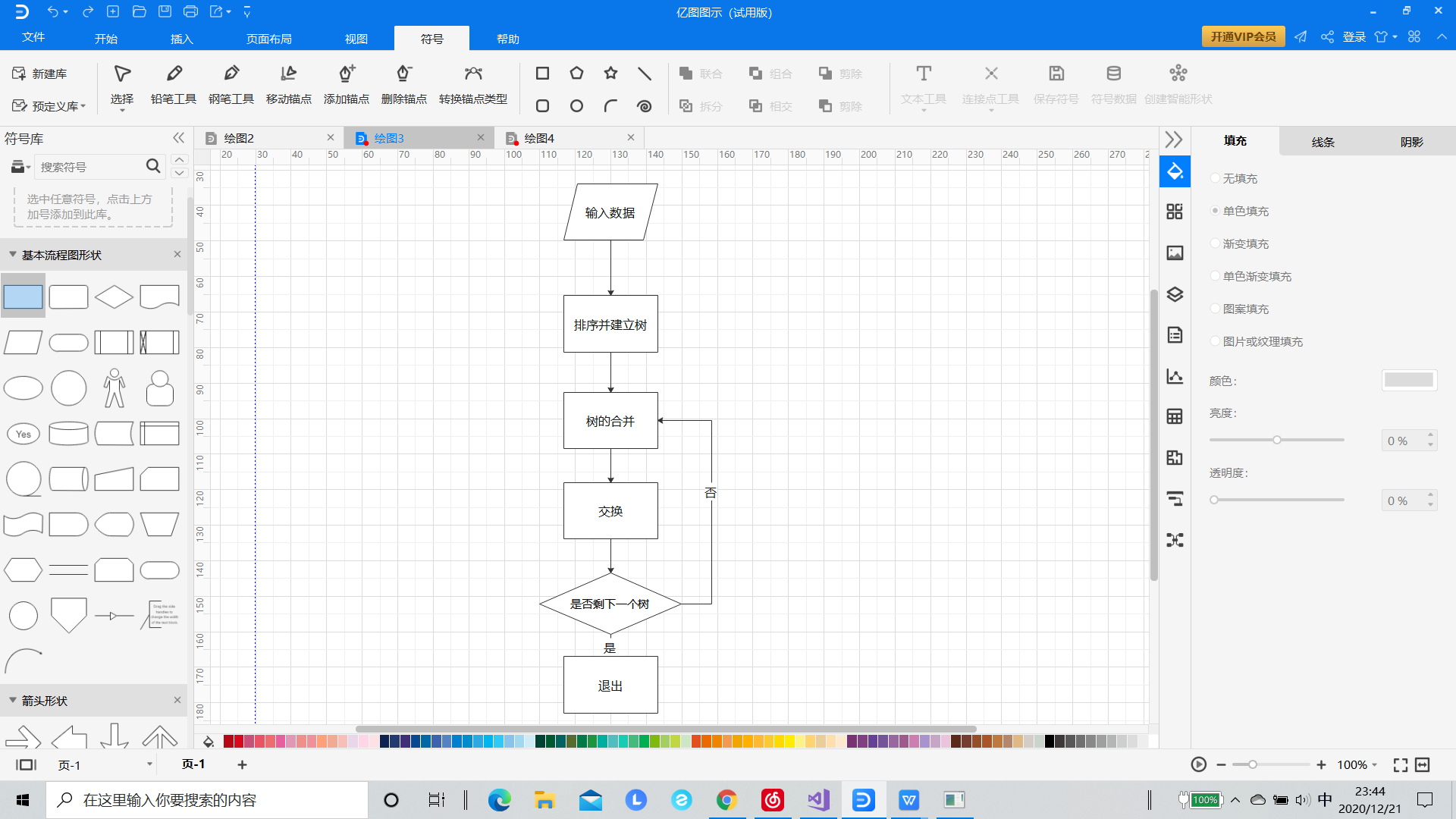
a = newone;

return 1;

### }}

# 3.7总体系统实现

### 3.7.1 总体系统流程图



### 3.7.2总体系统核心代码

int main()

{

int end = 0;

for (;;)

{

int putnumber[1024];

int number = 0;

//进行输入和排序，从小到大

cin >> number;

if (number <= 1) { cout << "输入的数小于或等于1 错误" << endl; }//边界条件

else

{

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

{

cin >> putnumber[n];

}

if (number == 2)

{

cout << putnumber[0] + putnumber[1] << endl;

}

else

{

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

{

int temp;

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

{

if (putnumber[m] > putnumber[m + 1])

{

temp = putnumber[m];

putnumber[m] = putnumber[m + 1];

putnumber[m + 1] = temp;

}

}

}

//通过从小到大的数组建立一个从小到大的链表来表示森林

huffman \*head, \*current;

head = new huffman;

current = new huffman(putnumber[0]);

head->link = current;

for (int n = 1; n < number; n++)

{

current->link = new huffman(putnumber[n]);

current = current->link;

}

huffman \*a;

for (;;)

{

head->link->combine(number, a);//最小的两项相加

head->link = a;

current = head;

for (int m = 1; m < number; m++)//进行交换

{

huffman \* a; huffman \* b;

if (head->link->link == NULL) { break; }

if (head->link->total <= head->link->link->total) { break; }

head->link->swap(a, b);

head->link = b;

head = b;

}

head = current;

if (number == 1) { break; }

}

int general = 0;

head->link->addtotal(general);//从根节点计算代价

cout << general << endl;

}

}

char answer; //循环输入

cout << "是否再次进行实验？ （Y，N）" << endl;

for (;;)

{

cin >> answer;

if (answer == 'Y' || answer == 'N') { break; }

cout << "请输入正确的字符" << endl;

}

if (answer == 'N') { break; }

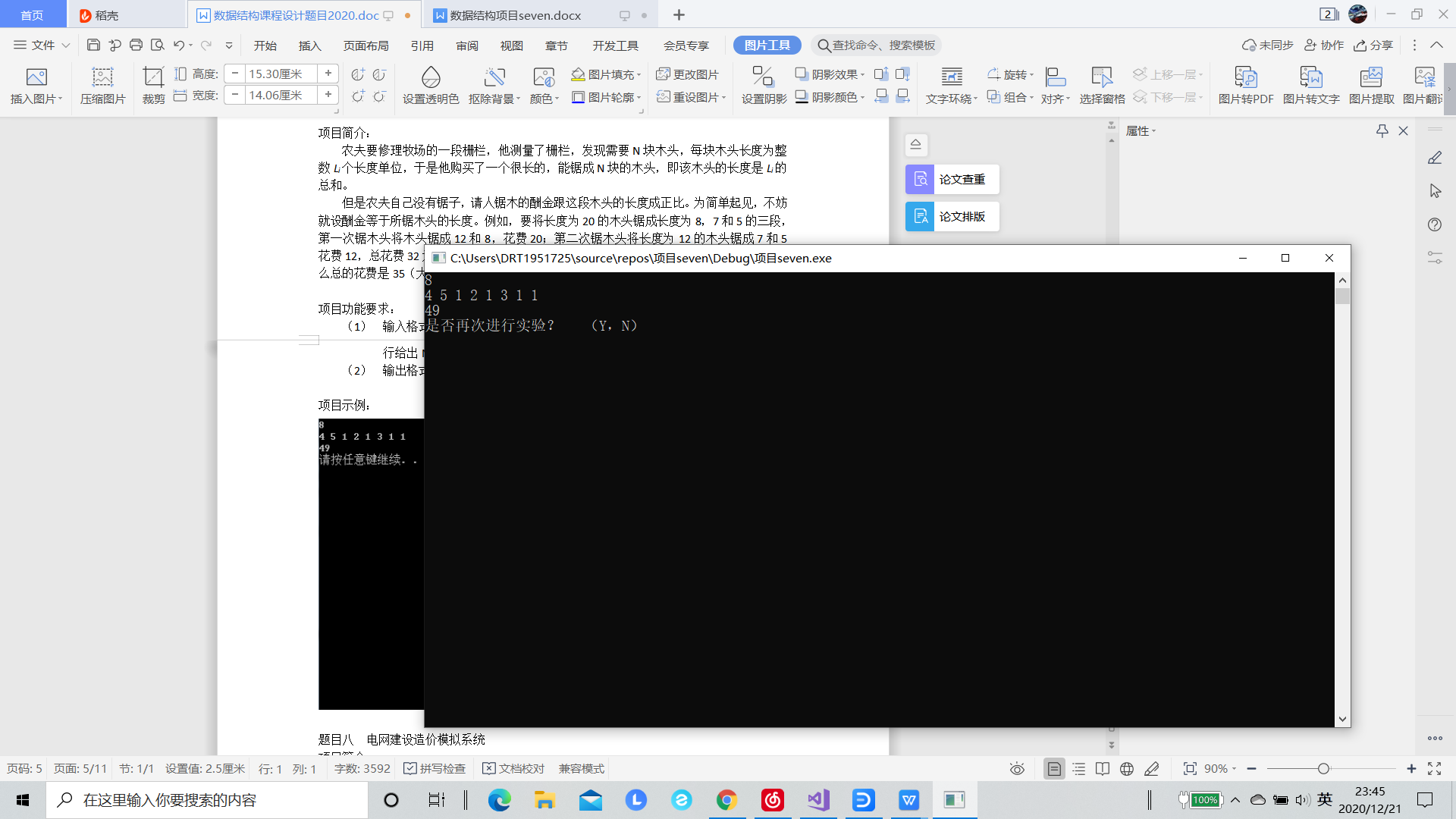
}

return 0;

}

# 4 测试

## 4.1 功能测试



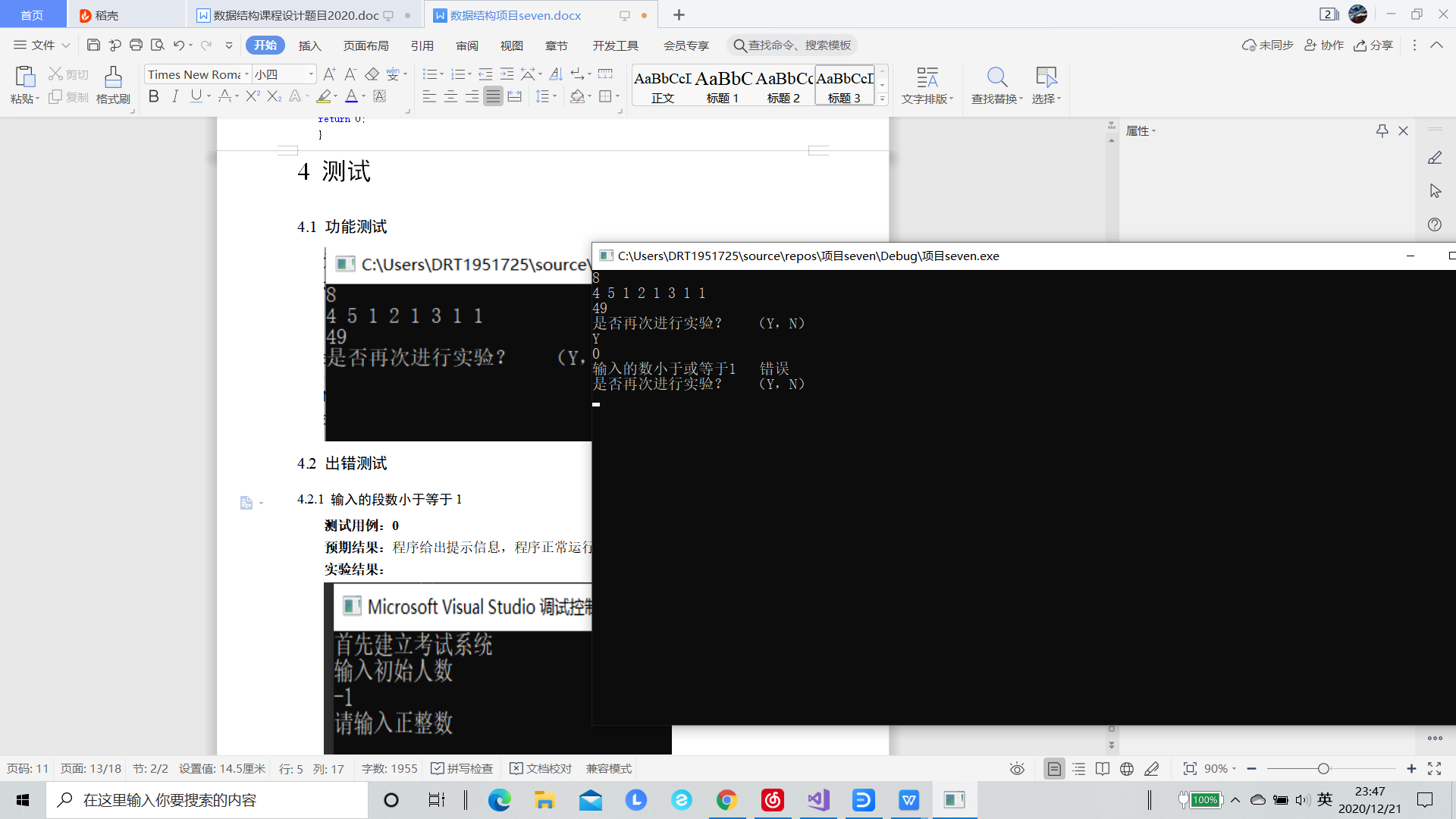
## 4.2 出错测试

### 4.2.1 输入的段数小于等于1

**测试用例：0**

**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：**



### 4.2.2 输入的数字小于1

**测试用例：8**

**-4 5 1 2 1 3 1 1**

**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：**

