项目说明文档

数据结构课程设计

——电网建设造价模拟系统

同济大学

Tongji University

目 录

[1 分析 1](#_Toc25765)

[1.1 背景分析 1](#_Toc16354)

[1.2 功能分析 1](#_Toc17259)

[2 设计 1](#_Toc11586)

[2.1 数据结构设计 1](#_Toc13782)

[2.2 成员与操作设计 1](#_Toc8063)

[2.3 系统设计 2](#_Toc5317)

[3 实现 2](#_Toc9266)

[3.1 添加顶点功能的实现 2](#_Toc31487)

[3.1.1 添加顶点功能流程图 2](#_Toc20818)

[3.1.2 添加顶点功能核心代码 2](#_Toc16470)

[3.1.3 添加顶点功能截屏示例 3](#_Toc22204)

[3.2 添加边功能的实现 5](#_Toc31976)

[3.2.1 添加边功能流程图 5](#_Toc21309)

[3.2.2 添加边功能核心代码 5](#_Toc21642)

[3.2.3 添加边功能截屏示例 6](#_Toc12655)

[3.3 构造最小树功能的实现 7](#_Toc27710)

[3.3.1 构造最小树功能流程图 7](#_Toc20169)

[3.3.2 构造最小树功能核心代码 7](#_Toc32750)

[3.3.3 构建最小树功能截图示例 9](#_Toc27934)

[3.4总体系统实现 11](#_Toc17123)

[3.4.1 总体系统流程图 11](#_Toc19459)

[4 测试 12](#_Toc10762)

[4.1 功能测试 12](#_Toc6582)

[4.1.1 添加顶点功能测试 12](#_Toc2971)

[4.1.2 添加边功能测试 12](#_Toc6301)

[4.1.3 构造最小生成树测试 13](#_Toc6454)

[4.1.4 展示最小生成树功能测试 14](#_Toc29052)

# 1 分析

## 1.1 背景分析

电网系统是生活中常见的一个系统，往往每个端点要和多个端点进行链接，可知图的结构更加符合其问题解决方案。我们可以通过一个邻接矩阵来图的表示。

## 1.2 功能分析

作为一个最简易的电网系统，我们应该所具有的功能是添加结点功能和添加边功能，构造最小树功能来计算造价，展现最小树功能来检验是否建立正确

# 

# 2 设计

## 2.1 数据结构设计

如上功能分析所述，该数据结构可以用一个邻接矩阵进行表示，其中每个矩阵的数据点都有其代价和是否以构建标志，在建立最小树的时候我们可以采用prime算法，每次取集合所有点链接外部的最小的边，并添加如对应的一个新结点，如此往复直到成功构建最小树停止

## 2.2 成员与操作设计

class matrix //邻接矩阵

{

public:

int cost;

int path;

};

class checkarray //用于检查是否所有的点都已经有边相连接

{

public:

int putin;

char name;

## };

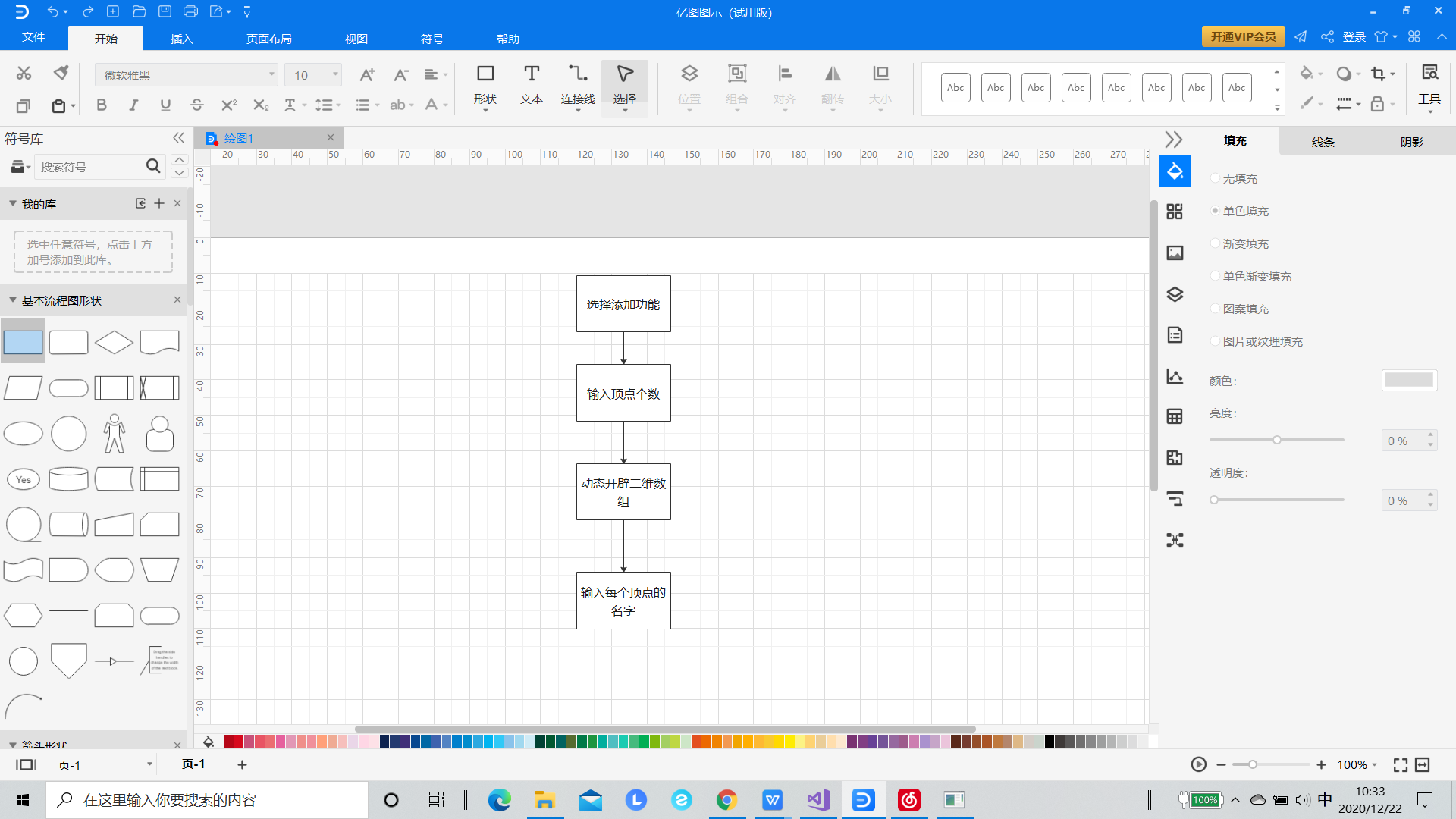
## 2.3 系统设计

系统首先调用添加功能实现对图的顶点和边的初始化，然后根据用户所输入的操作码生成对应的最小树，并进行展示。

# 3 实现

## 3.1 添加顶点功能的实现

### 3.1.1 添加顶点功能流程图



### 3.1.2 添加顶点功能核心代码

for (int n = 0; n < number; n++) //建立二维数组

{

a[n] = (matrix\*)malloc(number \* sizeof(matrix));

}

for (int n = 0; n < number; n++) //初始化

{

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

{

a[n][m].cost = 0;

a[n][m].path = 0;

}

}

for (int n = 0; n < number; n++) //将对角排除在外

{

a[n][n].cost = -1;

a[n][n].path = -1;

}

for (int n = 0; n < number; n++)//初始化

{

b[n].putin = 0;

b[n].name = '0';

}

char putinname;

cout << "请输入要输入的顶点名称（请按照小写字母顺序）" << " " << endl;

for (int n = 0; n < number; n++) //开始输入顶点的个数

{

cin >> putinname;

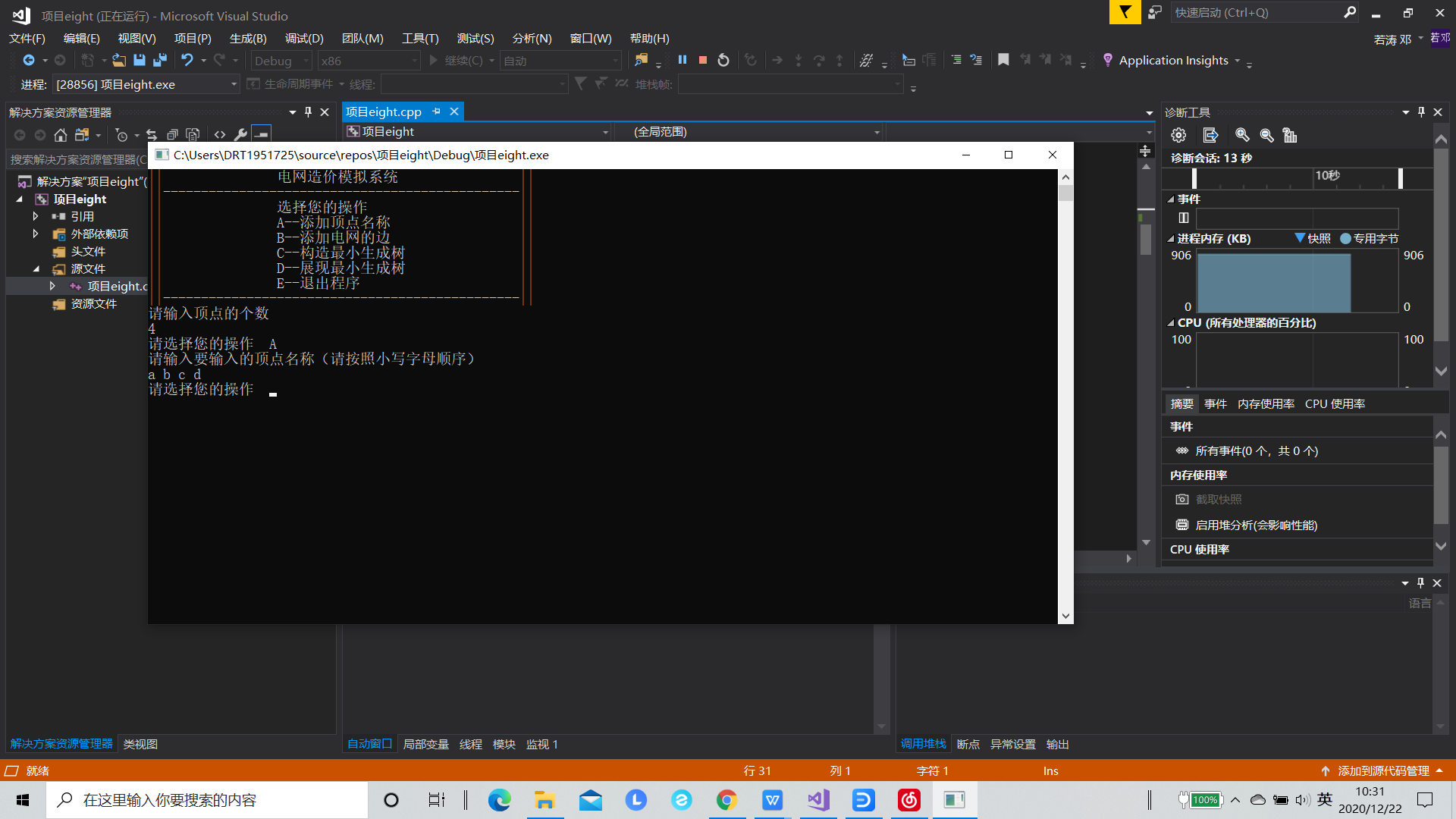
if (int(putinname) - 97 > number - 1) { cout << "输入的字母不符合字母表逻辑" << endl; break; }

if (b[int(putinname) - 97].name != '0') { cout << "重复输入" << endl; break; }

b[int(putinname) - 97].name = putinname;

}

### 3.1.3 添加顶点功能截屏示例



## 3.2 添加边功能的实现

### 3.2.1 添加边功能流程图

### 

### 3.2.2 添加边功能核心代码

for (;;) //输入各个顶点之间的路径长度

{

cin >> head;

cin >> tail;

cin >> length;

if (head == '?') { break; }

if (int(head) - 97 > number - 1) { cout << "head wrong" << endl; wrong = 1; break; }

if (int(tail) - 97 > number - 1) { cout << "tail wrong" << endl; wrong = 1; break; }

if (a[int(head) - 97][int(tail) - 97].cost != 0) { cout << "有重复的边出现" << endl; wrong = 1; break; }

a[int(head) - 97][int(tail) - 97].cost = length;

a[int(tail) - 97][int(head) - 97].cost = length;

b[int(head) - 97].putin = 1;

b[int(tail) - 97].putin = 1;

}

//边界错误情况判断

if (wrong == 1) { cout << "该步骤有错误，请重新进行该步骤" << endl; break; }

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

{

if (b[n].putin == 0) { cout << "有节点没有链接上去" << endl; wrong = 1; break; }

}

if (wrong == 1) { cout << "该步骤有错误，请重新进行该步骤" << endl; break; }

//将没有路径的顶点之间的路径封段

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

{

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

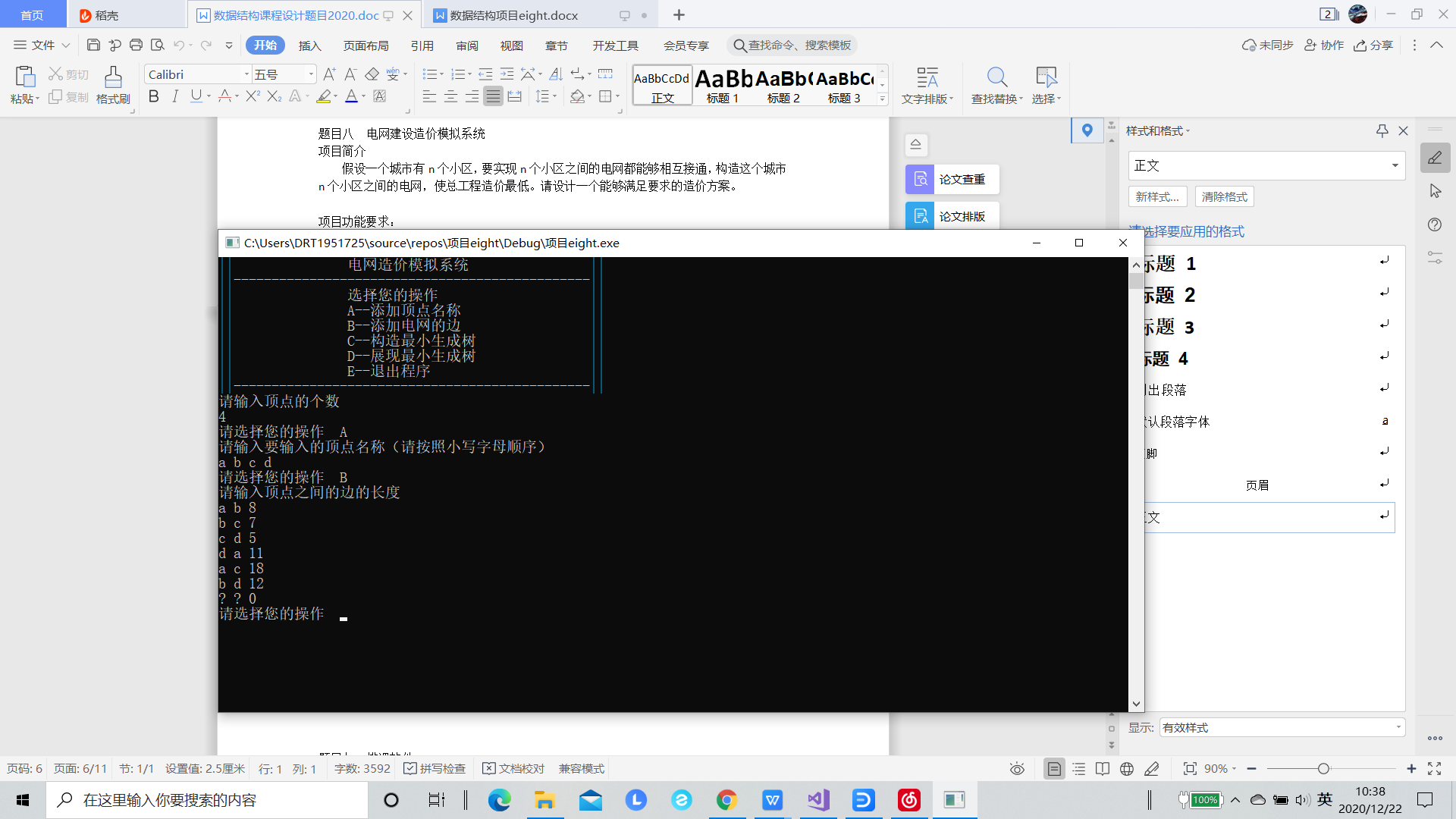
{

if (a[n][m].cost == 0) { a[n][m].path = -1; }

}

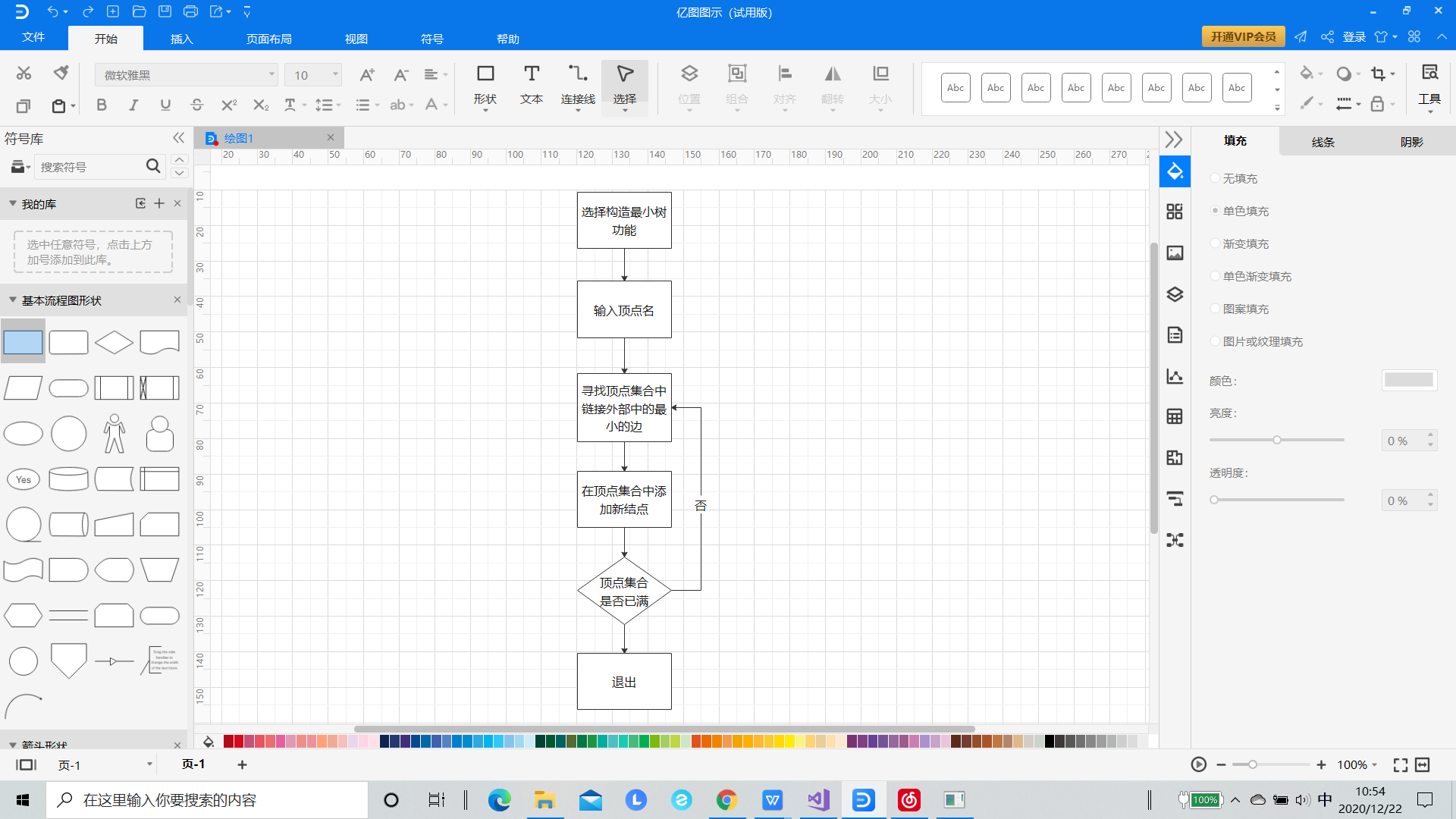
### }

### 3.2.3 添加边功能截屏示例



## 3.3 构造最小树功能的实现

### 3.3.1 构造最小树功能流程图



### 3.3.2 构造最小树功能核心代码

cout << "请输入顶点的名称" << " ";

char first;

cin >> first;

if (int(first) < int(b[0].name) || int(first) > int(b[number - 1].name))

{ cout << "输入的字符不在原先的顶点中" << endl; cout << "该步骤有错误，请重新进行该步骤" << endl; break; };

int row, col;

row = int(first) - 97;

int one = 0;

int temp = 0;

//搜索该确定行的最小的路径长度

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

{

int cost = 0; int stamp = 0;

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

{

if (a[row][n].path == 0)

{

cost = min(stamp, cost, a[row][n].cost);

if (stamp == 1)

{

stamp = 0;

col = n;

}

}

}

a[row][col].path = 1;

a[col][row].path = 1;

putin[one] = row;

//cout << putin[one] << endl;

one++;

putin[one] = col;

//cout << putin[one] << endl;

one++;

col = 0;

temp = 0;

//确定下一行从哪一行进行寻找

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

{

int x, y; x = y = 0;

x = find(a, putin[n], number);// cout << "x"<<x << endl;

y = find(a, putin[n + 1], number);// cout << "y" << y <<endl;

if (x == 0)

{

if (temp > y || temp == 0)

{

temp = y; row = putin[n + 1];

}

}

else

{

if (x < y)

{

if (temp > y || temp == 0)

{

temp = x; row = putin[n];

}

}

else

{

if (temp > y || temp == 0)

{

temp = y; row = putin[n + 1];

}

}

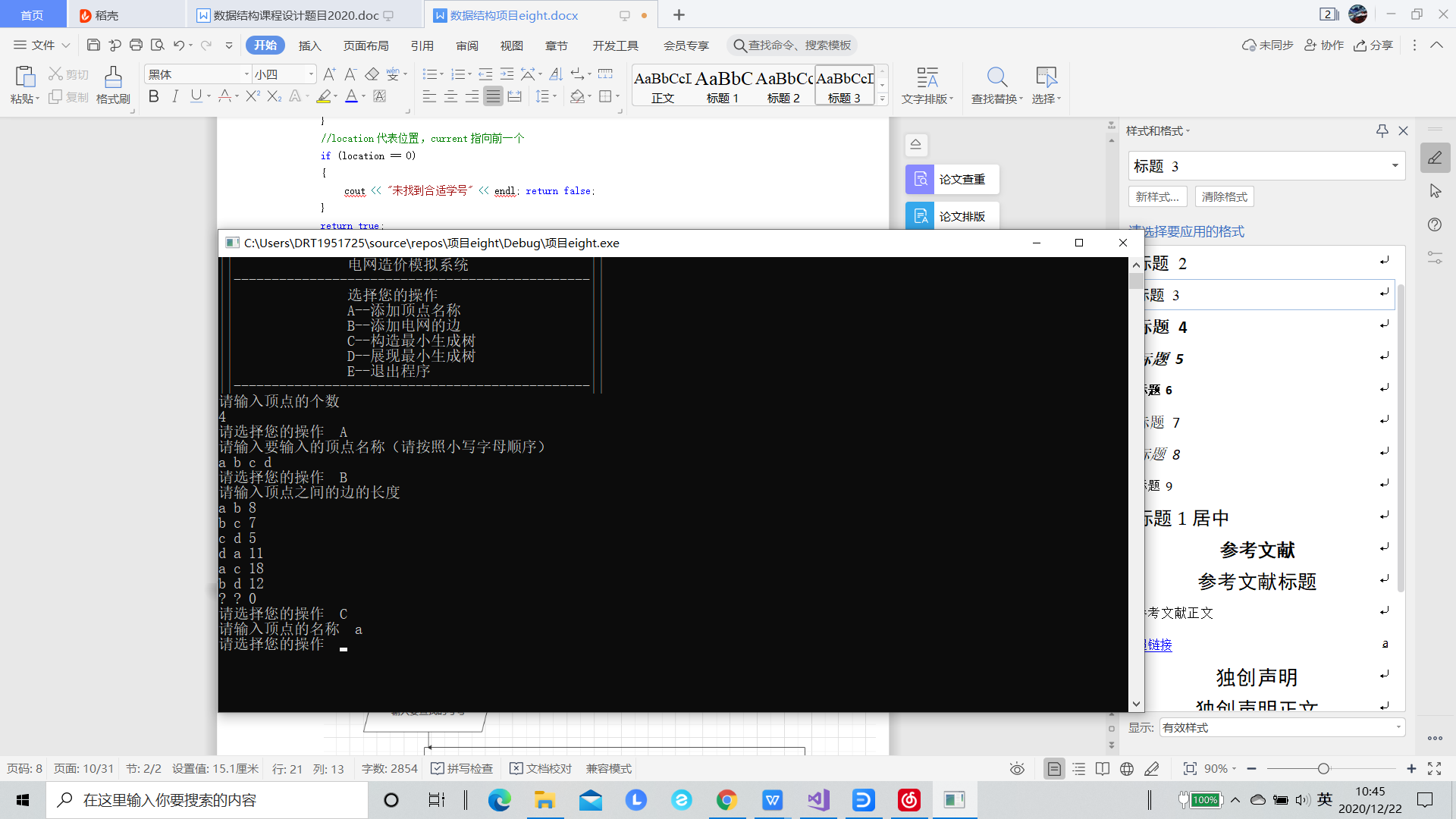
}

}

if (temp == 0) { cout << "wrong" << endl; break; }

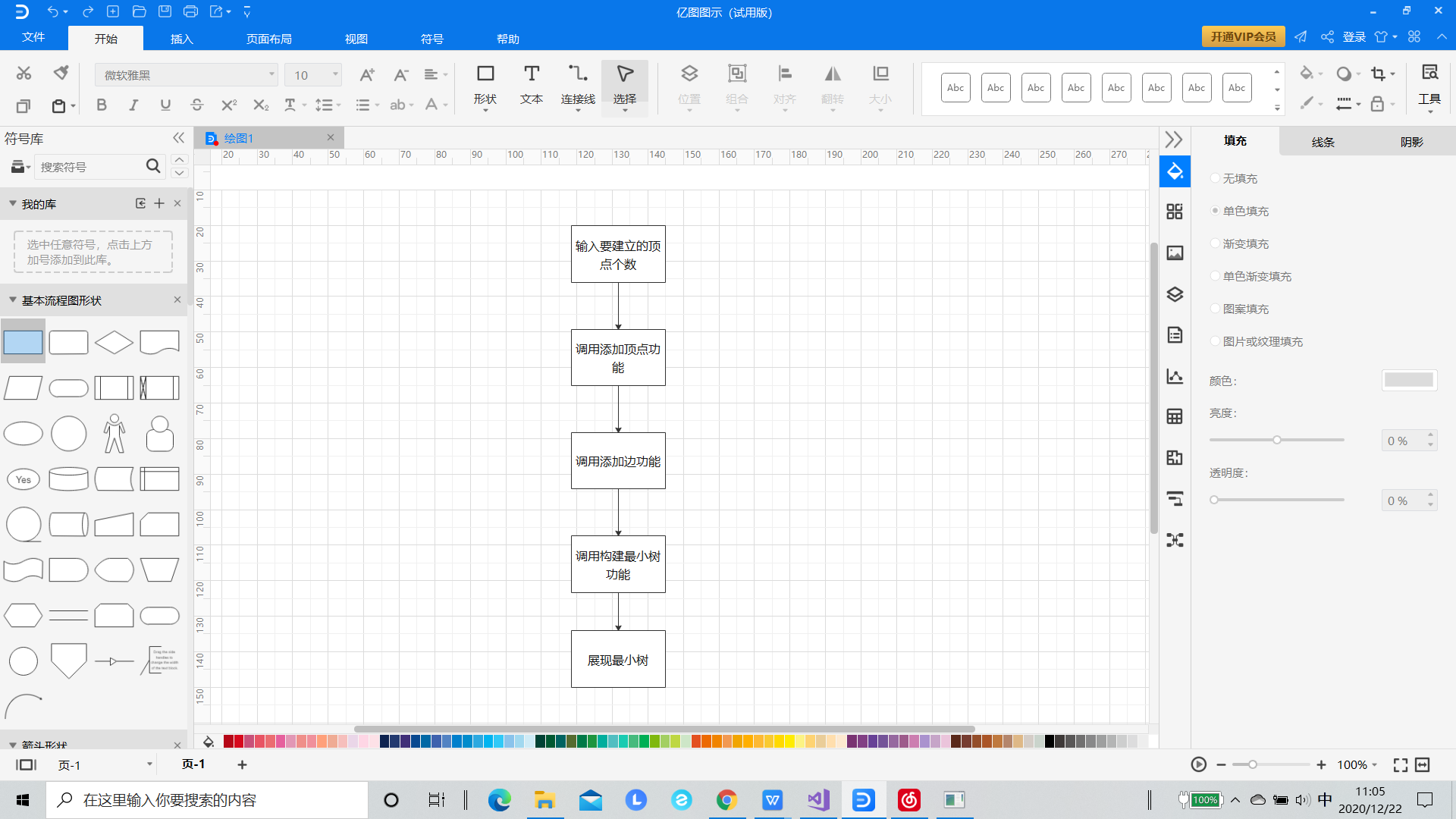
}

### 3.3.3 构建最小树功能截图示例



# 3.4总体系统实现

### 3.4.1 总体系统流程图



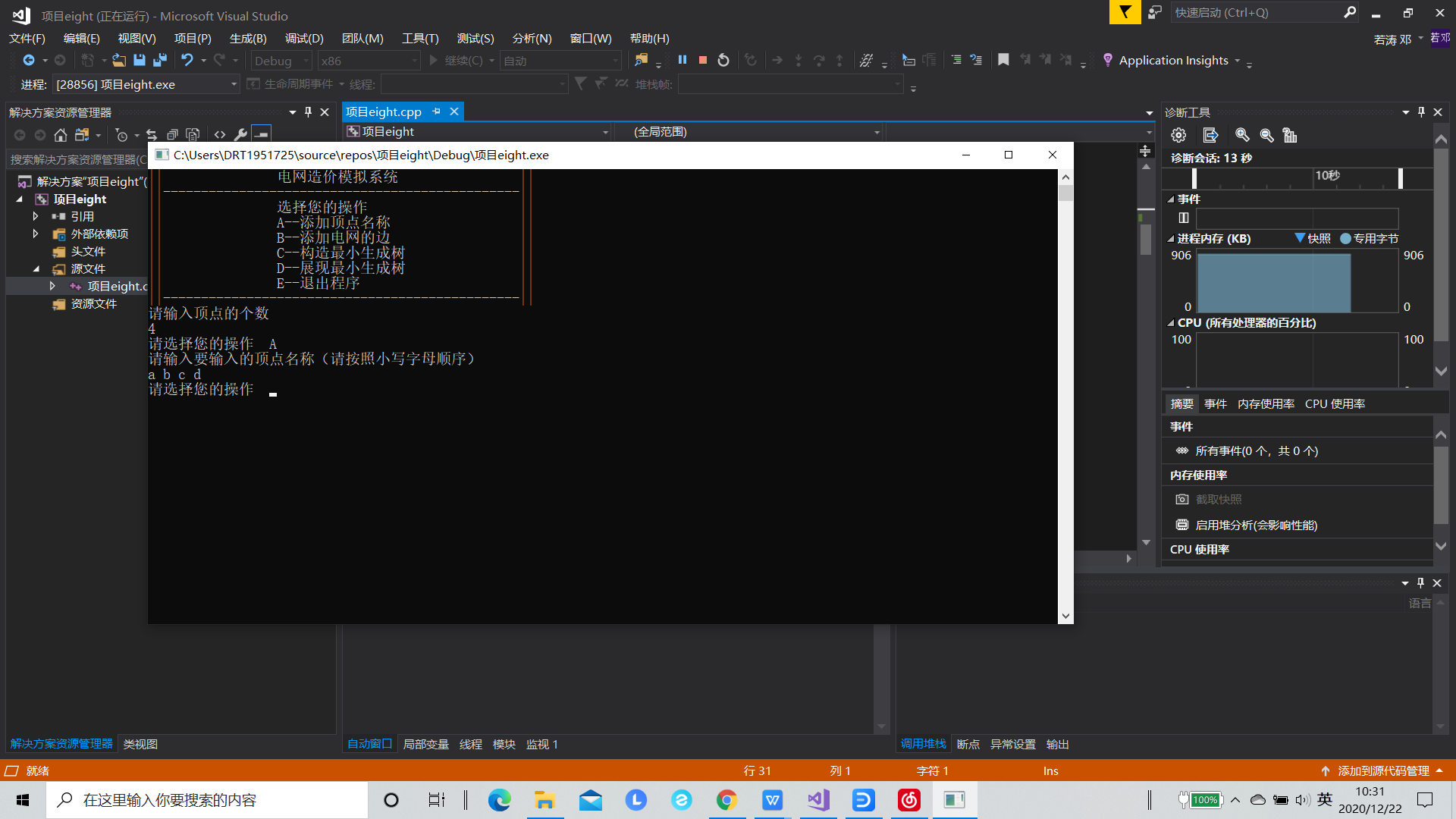
# 4 测试

## 4.1 功能测试

### 4.1.1 添加顶点功能测试

**测试用例**：4 a b c d

**实验结果**



### 4.1.2 添加边功能测试

**测试用例：**

**a b 8**

**b c 7**

**c d 5**

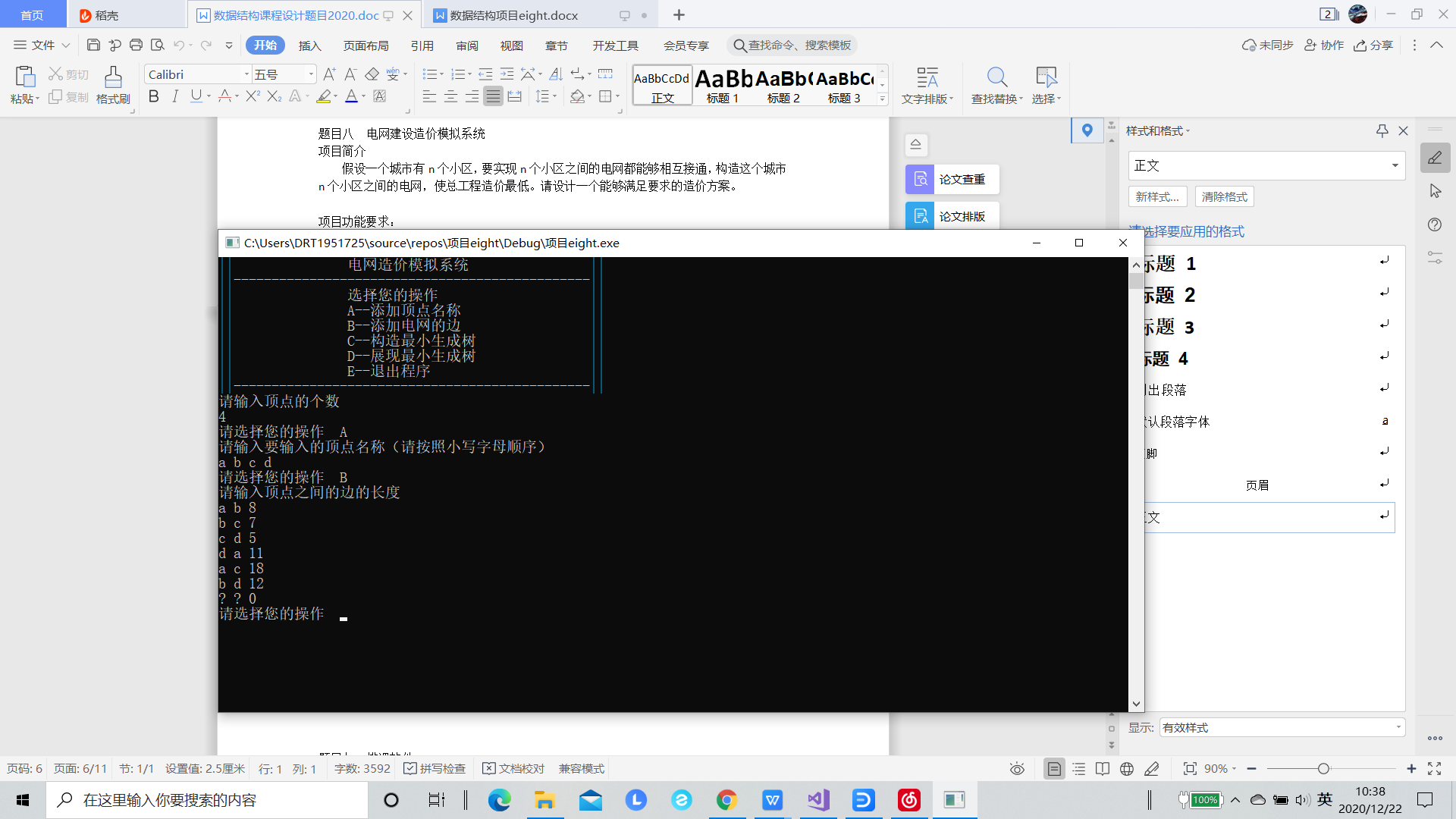
**d a 11**

**a c 18**

**b d 12**

**? ? 0**

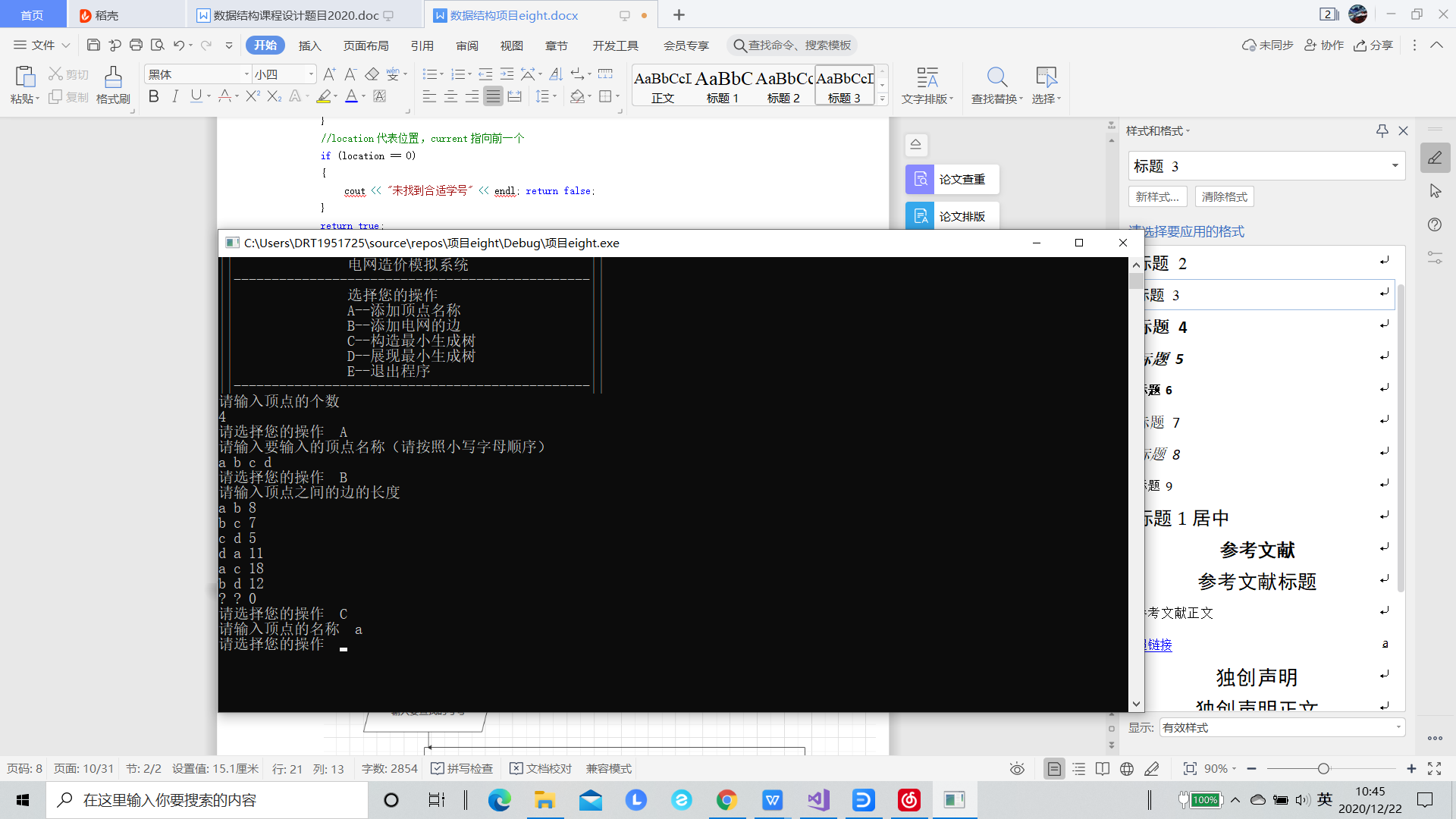
**实验结果：**



### 4.1.3 构造最小生成树测试

**测试用例：a**

**实验结果：**



### 4.1.4 展示最小生成树功能测试

**预期结果：**

**a 8 b b 7 c c 5 d**

**实验结果：**

