**《数据结构》课程设计总结**

****

**学 号 1651025**

**姓 名 汪 涵**

**专 业 计算机科学与技术**

**2018年8月**

目录

[第一部分 算法实现设计说明 3](#_Toc524351405)

[1.1 题目 3](#_Toc524351406)

[1.2 软件功能 3](#_Toc524351407)

[1.3 设计思想 3](#_Toc524351408)

[1.4 逻辑结构和物理结构 8](#_Toc524351409)

[1.5 开发平台 8](#_Toc524351410)

[1.6 系统的运行结果分析说明 8](#_Toc524351411)

[1.7 操作说明 12](#_Toc524351412)

[第二部分 综合应用设计说明 14](#_Toc524351413)

[2.1 题目 14](#_Toc524351414)

[2.2 软件功能 14](#_Toc524351415)

[2.3 设计思想 14](#_Toc524351416)

[2.4 逻辑结构与物理结构 19](#_Toc524351417)

[2.5 开发平台 19](#_Toc524351418)

[2.6 系统的运行结果分析说明 20](#_Toc524351419)

[2.7 操作说明 23](#_Toc524351420)

[第三部分 实践总结 25](#_Toc524351421)

[3.1 所做的工作 25](#_Toc524351422)

[3.2 总结与收获 25](#_Toc524351423)

[第四部分 参考文献 25](#_Toc524351424)

# 第一部分 算法实现设计说明

## 题目

二叉树，完成：

1. 建立一棵二叉树，并对它进行先序、中序、后序遍历。
2. 统计树中的叶子节点个数。
3. 分别对它进行先序、中序、后序线索化。
4. 实现先序、中序线索树的遍历。
5. 显示该树和线索化后的（此要求可视情况选择是否完成）。

## 软件功能

根据题目的要求，我所设计的软件应具有以下功能：

* 可以接受用户输入需要构建的二叉树的先序遍历。
* 根据用户输入的先序遍历构建二叉树并图形化显示在界面上。
* 在构建二叉树时统计其叶子节点个数并输出。
* 可以让用户选择对二叉树进行先序、中序、后序遍历，并在界面上动态显示遍历过程，同时输出遍历结果。
* 在构建完二叉树后允许用户点击按钮来对二叉树进行先序或中序线索化，并实现先序或中序线索树的遍历，并输出相应遍历结果以及各结点的前驱和后继分别是Link(连接)或Thread(线索)。
* 在对二叉树线索化吼将线索图形化显示在界面上（此功能在本软件中尚未实现）。

这些功能的实现方式在详见1.3设计思想。

## 设计思想

本软件设计思路基本如下，在数据结构与算法这一部分，软件的实现思路及算法基本流程如下：

* 根据用户所给出的二叉树先序遍历，用BiTree.js中的Create\_BiTree()函数递归构建二叉树。并且在递归回溯过程中，对每一个构建好其子树的结点进行判断，如果其左右孩子结点均为空，则该结点为叶子结点，统计变量Leaf\_Num加1。
* 对于二叉树先序、中序、后序遍历均使用递归算法，分别通过BiTree.js中的Pre\_Order\_Traverse()、In\_Order\_Traverse()、Post\_Order\_Traverse()函数实现。其中核心部分为将当前节点加入遍历结果中的顺序，先序遍历是先将当前结点加入遍历结果，再依次对该结点的左右孩子进行递归操作，中序遍历是先对该结点的左孩子进行递归操作，然后将当前结点加入遍历结果，最后对该结点的右孩子进行递归操作，后序遍历是先依次对该结点的左右孩子进行递归操作，再将当前结点加入到遍历结果。最后将遍历结果输出。

关键代码：

先序：

pre\_seq.push(T.data);

pre\_order\_traverse(T.left);

pre\_order\_traverse(T.right);

中序：

in\_order\_traverse(T.left);

in\_seq.push(T.data);

in\_order\_traverse(T.right);

后序：

post\_order\_traverse(T.left);

post\_order\_traverse(T.right);

post\_seq.push(T.data);

* 对于二叉树的先序、中序线索化均使用递归算法，分别通过BiTree.js中的Pre\_Order\_Threading()、Post\_Order\_Threading()函数实现。其中核心部分是递归操作和对当前节点构建线索的顺序，先序线索化是先对当前结点构建线索，然后再对当前结点的左右孩子进行递归操作（在前驱/后继为Link时），中序线索化是先对当前结点的左孩子进行递归操作（在前驱为Link时），再对当前结点构建线索，最后再对当前结点的右孩子进行递归操作（在后继为Link时）。

关键代码：

先序：

if (!T.left) {

T.left = Prev;

T.LTag = "Thread";

}

if (Prev && !Prev.right) {

Prev.right = T;

Prev.RTag = "Thread";

}

Prev = T;

if (T.LTag === "Link") {

pre\_order\_threading(T.left);

}

if (T.RTag === "Link") {

pre\_order\_threading(T.right);

}

中序：

if (T.LTag === "Link") {

in\_order\_threading(T.left);

}

if (!T.left) {

T.left = Prev;

T.LTag = "Thread";

}

if (Prev && !Prev.right) {

Prev.right = T;

Prev.RTag = "Thread";

}

Prev = T;

if (T.RTag == "Link") {

in\_order\_threading(T.right);

}

* 对于先序、中序线索树的遍历的核心则是循环寻找下一个遍历的结点。

关键代码：

先序:

while (T) {

while (T.left && T.LTag === "Link") {

pre\_seq.push(T.data);

T = T.left;

}

pre\_seq.push(T.data);

if (T.LTag === "Thread") {

T = T.right;

}

while (T) {

if (T.left && T.LTag == "Link") {

break;

}

pre\_seq.push(T.data);

T = T.right;

}

}

中序：

while (T) {

while (T.LTag === "Link") {

T = T.left;

}

in\_seq.push(T.data);

while (T.right && T.RTag === "Thread") {

T = T.right;

in\_seq.push(T.data);

}

T = T.right;

}

另外，在实际实现中，还需要将每个结点的前驱和后继以及类型加入到遍历结果中，以此来展示线索化的结果。

本软件在可视化方面的实现思路及基本流程基本如下：

除去HTML以及CSS的整体界面布局等，二叉树的可视化显示和遍历的动态显示是关键部分。此处使用SVG实现二叉树的可视化以及遍历的动态显示。先对二叉树进行层次遍历，在层次遍历中，保存每个结点的相关信息，具体如下：

data: T.data, //该结点数据

floor: i, //该结点处于的深度

widthRatio: 1 / Math.pow(2, i - 1), //该结点所占画布比例

which: which //该结点是父亲节点的左孩子(0)或右孩子(2)

遍历结束后将遍历结果传给Show\_Tree\_Traverse()函数，在本函数中，先通过树的深度和最深层结点的展开计算所需画布的大小。然后通过计算各个结点的位置画出整个二叉树，关键代码如下：

//父节点宽度的最左边

var start = father.cx - width \* father.widthRatio / 2;

//当前结点的横坐标

cx = start + width \* nodes[i].widthRatio \* (nodes[i].which + 0.5);

console.log(start, cx);

cy = nodes[i].floor \* 80;

nodes[i].cx = cx;

nodes[i].cy = cy;

//画线

lineStr += '<line x1="' + cx + '" y1="' + cy + '" x2="' + nodes[i].father.cx +

'" y2="' + nodes[i].father.cy + '" style="stroke:black;stroke-width:2" />';

console.log(lineStr);

//画结点

circleStr += '<circle cx="' + cx + '" cy="' + cy + '" r="20" fill="' + nodeColor + '"/></circle>';

//调整文本缩进

var textcx = nodes[i].data > 9 ? (nodes[i].cx - 10) : (nodes[i].cx - 5),

textcy = (nodes[i].cy + 6);

textStr += '<text x="' + textcx + '" y="' + textcy + '" fill="white">' + nodes[i].data + '</text>';

动态显示遍历过程则是根据遍历结果数组，遍历该数组，每次都调用Show\_Tree\_Traverse()函数，其中对当前遍历结点换种颜色显示，其余结点颜色不更换。关键代码如下：

for (var i = 0; i <= traversalArr.length; i++) {

setTimeout(function (i) {

Show\_Tree\_Traverse(Tree, traversalArr[i], nodes);

if (i >= traversalArr.length) {

button.click = true;

console.log("OK");

var result = "";

for (var i = 0; i < traversalArr.length; i++) {

result = result + traversalArr[i];

}

console.log(result);

switch (index) {

case 1: $("#output").append("<span><strong>先序遍历结果：%data%</strong></span>".replace("%data%", result));

break;

case 2: $("#output").append("<span><strong>中序遍历结果：%data%</strong></span>".replace("%data%", result));

break;

case 3: $("#output").append("<span><strong>后序遍历结果：%data%</strong></span>".replace("%data%", result));

break;

}

}

}, 400 \* i, i);

}

除了以上的实现思路和基本流程外，关于点击按钮触发事件等实现都较为基础，均为调用函数等操作，在此不予赘述。

## 逻辑结构和物理结构

本题是二叉树的相关操作，故本程序中应用的逻辑结构为非线性树状结构，物理结构如下：

function Node(data) {

this.data = data;

this.left = null; //左孩子

this.right = null; //右孩子

this.LTag = "Link"; //左标记

this.RTag = "Link"; //右标记

}

## 开发平台

* 程序开发平台：Visual Studio Code、Chrome
* HTML5、CSS、Javascript
* JQuery(JavaScript框架、<http://jquery.com/>)
* Bulma(CSS框架、

<https://cdnjs.cloudflare.com/ajax/libs/bulma/0.6.2/css/bulma.min.css>)

* Exe可执行文件使用Electron打包
* 运行环境：任意支持HTML5的浏览器（如Chrome、Firefox等）打开index.html或者Windows系统的电脑打开App.exe。（注意：由于在线调用了第三方框架，在运行时要保持全程联网）

## 系统的运行结果分析说明

在开发过程中使用Visual Studio Code进行开发。

图片包含 屏幕截图, 电子产品, 陈列, 计算机

已生成极高可信度的说明

在调试过程中使用Chrome开发者工具进行调试。

图片包含 屏幕截图

已生成极高可信度的说明

在开发过程中，遇到一些问题，但基本上都是在可视化方面的问题，例如界面布局，重复显示文字等问题。没有遇到较大问题，在此举一例说明：

我发现，在多次点击确认输入按钮时，下发文字“叶子节点个数为：”总是会重复出现，经过调试，我发现应该在每次点击按钮后将id=”leafnum”的div元素内部所有元素全部清空即可。

本程序运行正确、稳定，能正确显示叶子节点数，可视化二叉树，动态显示遍历过程以及展现线索化结果。而且在可以在不刷新页面的情况下输入新的数据，进行新的操作。

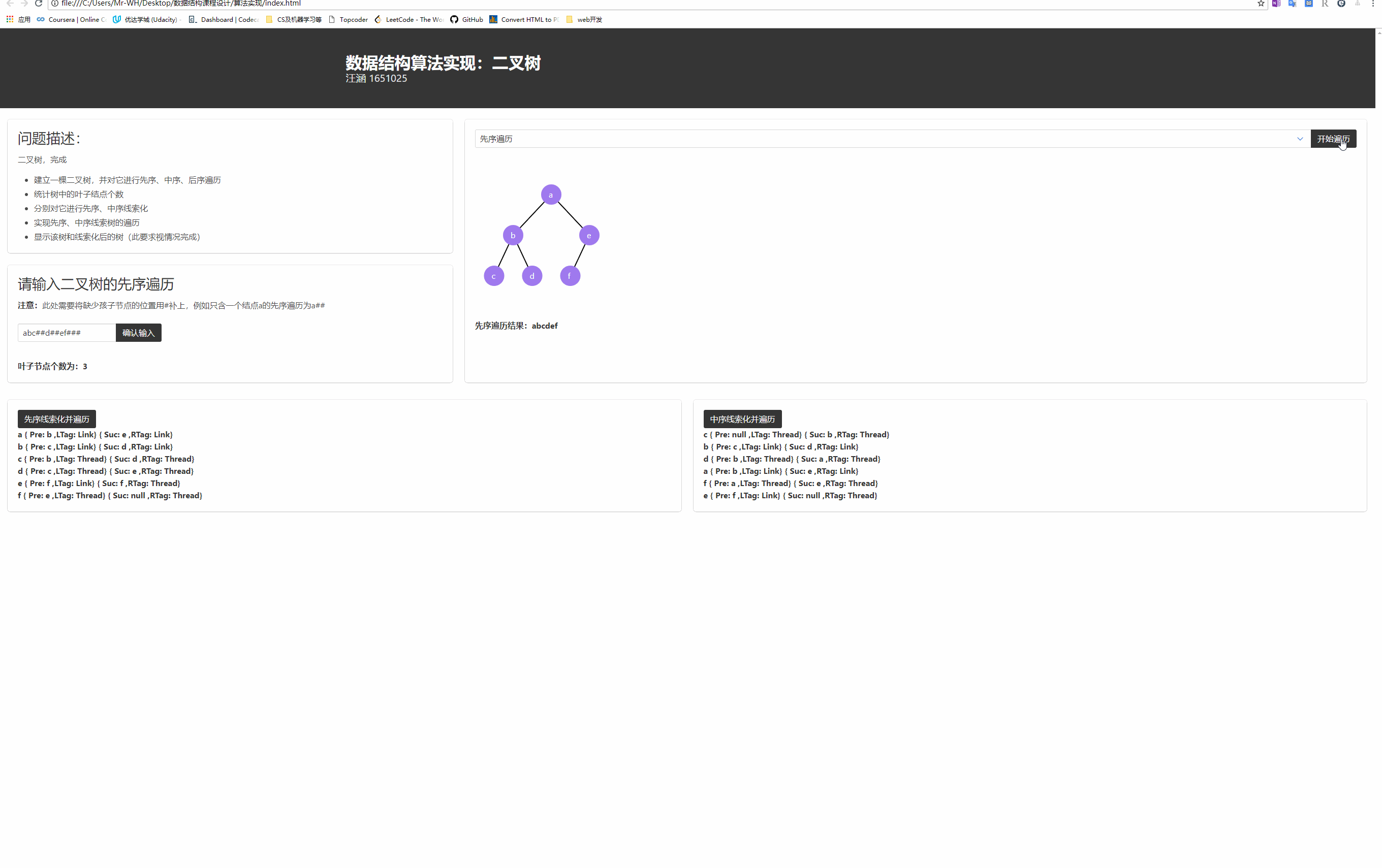
具体运行结果：

输入二叉树的先序遍历abc##d##ef###，点击确认输入后，再点击先序/后序线索化并遍历按钮后，会出现如下运行结果：

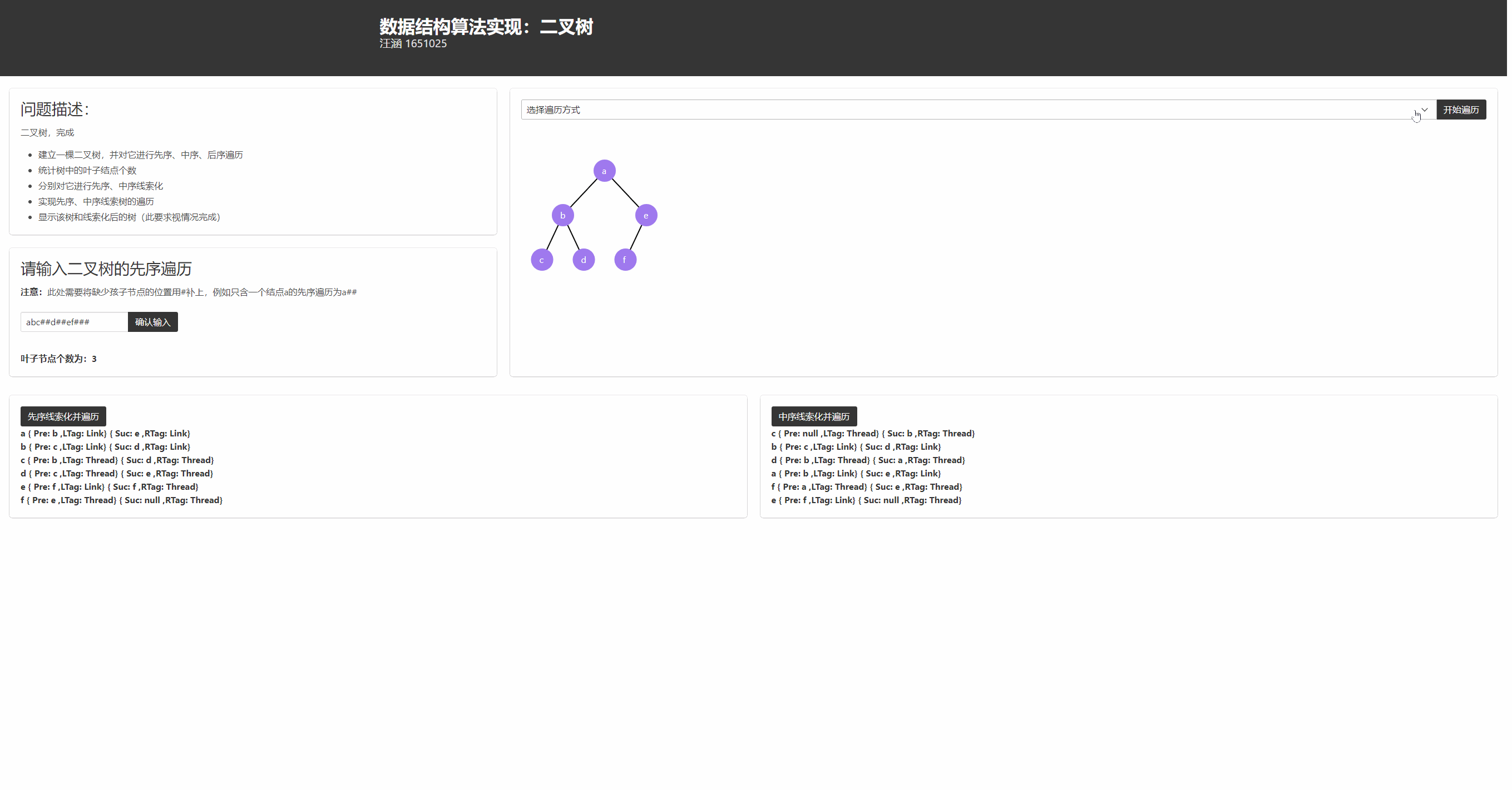
图片包含 屏幕截图

已生成极高可信度的说明

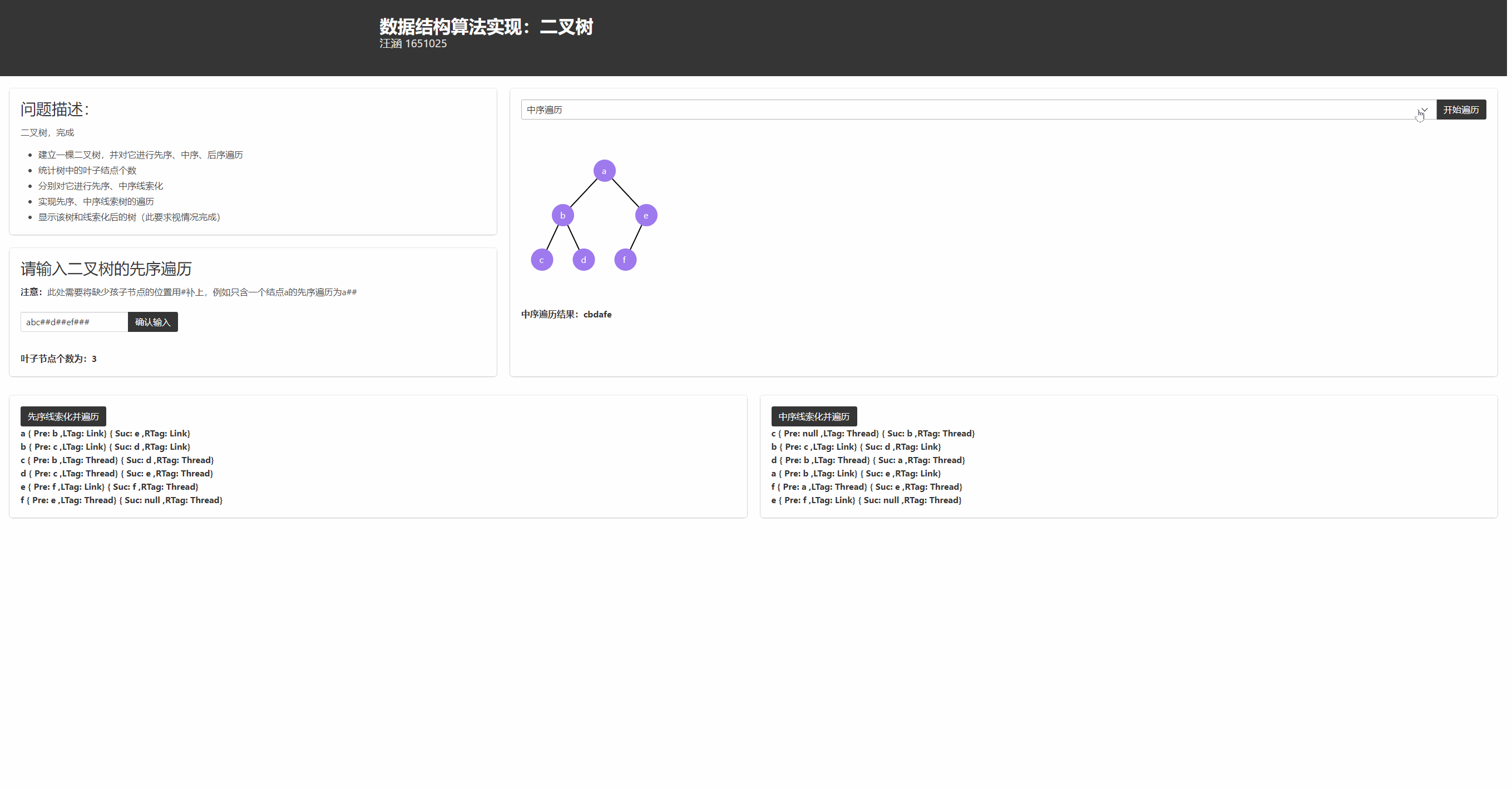
选择先序遍历，点击遍历后，会出现如下运行结果（按住ctrl并单击图片可访问相应gif链接）：

[](运行效果截图等/二叉树1.gif)

选择中序遍历，点击遍历后，会出现如下运行结果（按住ctrl并单击图片可访问相应gif链接）：

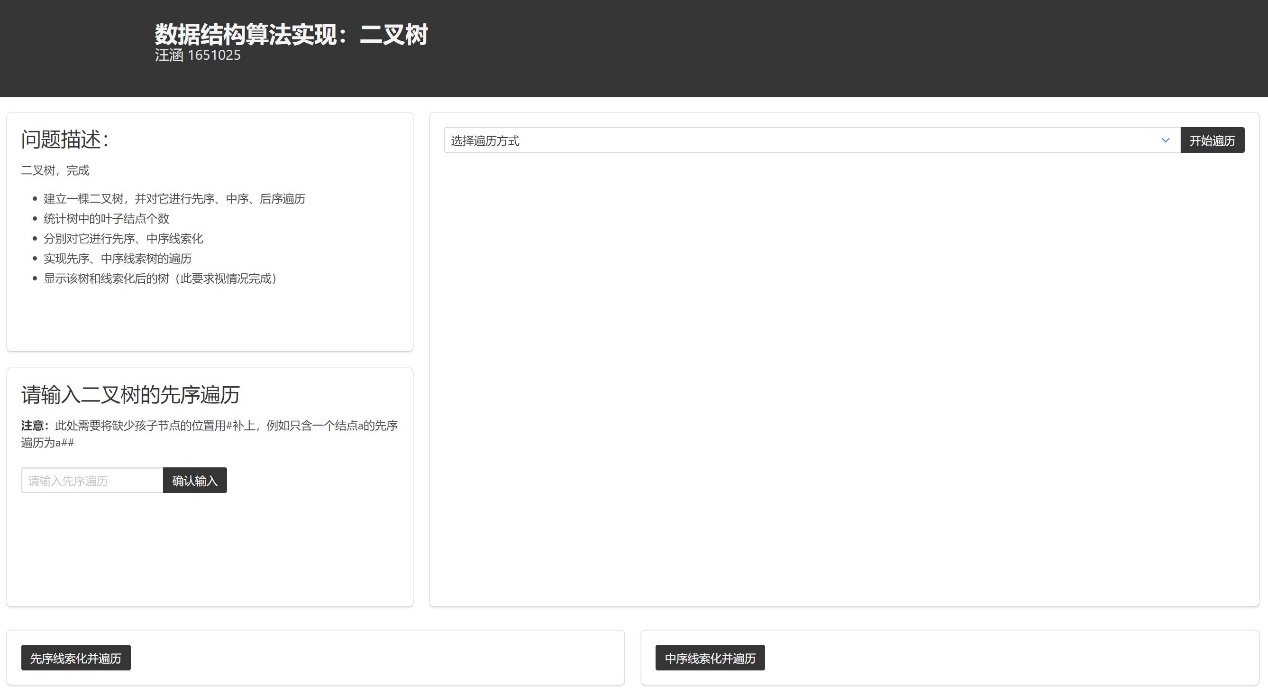
[](运行效果截图等/二叉树2.gif)

选择后序遍历，点击遍历后，会出现如下运行结果（按住ctrl并单击图片可访问相应gif链接）：

[](运行效果截图等/二叉树3.gif)

## 操作说明

* 双击打开index.html或App.exe，将会进入如下界面：



* 在输入框中输入二叉树的先序遍历（如：abc##d##ed###），然后点击确认输入，将会进入如下界面：

图片包含 屏幕截图

已生成极高可信度的说明

* 选择遍历方式，点击开始遍历，将会进入如下界面：

图片包含 屏幕截图

已生成极高可信度的说明

* 点击先序/中序线索化并遍历按钮，将会进入如下界面：

图片包含 屏幕截图

已生成极高可信度的说明

* 可以在输入框中输入新一组数据，然后按照上述步骤进行新的操作。

# 第二部分 综合应用设计说明

## 2.1 题目

在某社会关系网络中，一个人属性包括所在地区、就读的各级学校、工作单位等，每一个人有众多好友，并可以根据个人兴趣及社会活动加入到某些群组。现需设计一算法，从该社会网络中某一人出发，寻找其可能认识的人。例如根据两个人共同好友的数量及所在群组情况，来发现可能认识的人；通过就读的学校情况发现可能认识的同学。

1. 通过图形化界面，显示某一人的社会关系网络。
2. 寻找某一人可能认识的人（不是其好友），并查看这些人与其关联度（共同好友数）。
3. 根据可能认识的关联度对这些人进行排序。

## 2.2 软件功能

根据题目要求，我所设计的软件应具有以下功能：

* 可以接受用户不断输入添加一个人的姓名及其属性。
* 可以图形化显示整个社会关系网络。
* 可以根据用户的不断输入，实时更新显示社会关系网络并图形化显示。
* 用户可以将光标移动到社会关系网络中代表某人的节点上，此时显示其与他人的社会关系。
* 用户可以双击社会关系网络中代表某人的节点上，此时高亮显示该人的社会关系网络，再次双击取消。
* 用户可以输入某一个的姓名来查找他可能认识的人（非好友），并根据关联度对可能认识的人进行排序。

这些功能的实现方式在详见2.3设计思想。

## 2.3 设计思想

本软件设计思路基本如下，在数据结构与算法这一部分，软件的实现思路及算法基本流程如下：

* 根据社会关系网络的特性，我选择无向图作为具体数据结构，每个节点表示每个人，两个节点间的边则表示两个人之间的关系，并使用邻接表来存储。
* 在处理每个新添加的人与其它人的关系时，我是将遍历节点存储数组，将其中每个人与新添加的人进行比较，比较两者各项属性的异同，根据此添加边，两者相同属性越多，则两者的社会关系越近，两节点间的边的权重则越小。

关键代码实现如下：

var flag = false;

var edge = {

"source": nodes.length - 1, //边的起点

"target": i, //边的终点

"relation": "", //边所代表的关系

"value": 3 //边的权重，即两人社会关系的远近

};

console.log(edge);

if (nodes[i].location === nodes[nodes.length - 1].location) {

flag = true;

edge.relation = edge.relation + "同城";

edge.value = edge.value - 0.5;

}

if (nodes[i].school1 === nodes[nodes.length - 1].school1 || nodes[i].school2 === nodes[nodes.length - 1].school2 || nodes[i].school3 === nodes[nodes.length - 1].school3) {

flag = true;

if (edge.relation === "") {

edge.relation = edge.relation + "同学";

}

else {

edge.relation = edge.relation + "/同学";

}

if (nodes[i].school1 === nodes[nodes.length - 1].school1) {

edge.value = edge.value - 0.5;

}

if (nodes[i].school2 === nodes[nodes.length - 1].school2) {

edge.value = edge.value - 0.5;

}

if (nodes[i].school3 === nodes[nodes.length - 1].school3) {

edge.value = edge.value - 0.5;

}

}

if (nodes[i].workplace === nodes[nodes.length - 1].workplace) {

flag = true;

if (edge.relation === "") {

edge.relation = edge.relation + "同事";

}

else {

edge.relation = edge.relation + "/同事";

}

edge.value = edge.value - 0.5;

}

if (flag) {

edges.push(edge);

}

* 在查找某人可能认识的人（非好友）的过程中，遍历其余所有节点，当该节点在邻接表中与查找点没有边相连，则将该节点作为可能认识的人的候选，此时，需要遍历查找点和该结点在邻接表中相邻的点，共同的相邻点越多，即两人的共同好友越多，则关联度越大。

关键算法实现如下：

var may\_know\_people = [];

for (var i = 0; i < Adj.length; i++) {

if (i != loc) {

if (Adj[i].indexOf(loc) === -1) {

var num = 0;

for (var j = 0; j < Adj[i].length; j++) {

if (Adj[loc].indexOf(Adj[i][j]) != -1) {

num++;

}

}

if (num != 0) {

var people = {

"name": nodes[i].name,

"correlation": num

};

may\_know\_people.push(people);

}

}

}

}

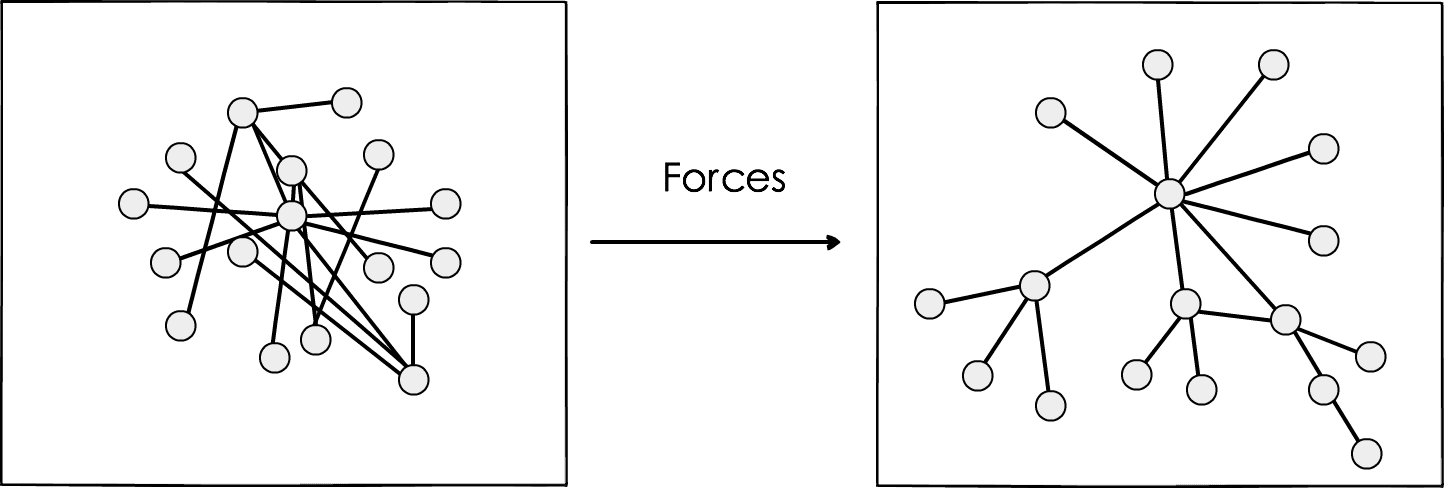
本软件在可视化方面的实现思路及基本流程基本如下：

除去HTML以及CSS的整体界面布局等，社会关系网络的可视化显示和交互是关键部分。

* 此次使用了D3.js可视化数据工具来实现社会网络关系的可视化与交互。其中力导向图十分适合渲染关系型信息图。

先来简单介绍一下什么是力导向图:

我们可以把整张 Network 想象成一个物理仿真系统(Simulation)。系统中的每个节点(Node)都是一个带有能量的粒子，粒子与粒子之间存在斥力（如模拟[库伦斥力](https://link.jianshu.com?t=http%3A%2F%2Fwww.baike.com%2Fwiki%2F%25E5%25BA%2593%25E4%25BB%2591%25E5%25AE%259A%25E5%25BE%258B" \t "_blank)），而被边(Link)所连结的粒子受到牵引力（如模拟[胡克弹力](https://link.jianshu.com?t=http%3A%2F%2Fbaike.baidu.com%2Flink%3Furl%3D_zEu0KBXE5_6FCcvJnwLXdvPcuor0MKZtRTe9KSjEzbF_fZvhsHwnQJgK7JKvpUmb4jm5AjAU6aGN8TClXOMty5FyD3bcq0L5YqZLEdbXTVMolTYpeN5iC40cYuSQl3W" \t "_blank)）。系统中的粒子在斥力和引力的不断作用下，从随机无序的布局(Layout)不断发生位移，逐渐趋于平衡有序的布局。同时整个仿真系统的能量也在不断消耗，经过数次迭代后，粒子之间不再发生相对位移，整个系统达到最终稳定平衡的状态。



* 在构建力导向图的过程中，我将节点全部存放在数组nodes[]中，每个节点包括该人的姓名及各项属性，将边全部存放在数组edges[]中，每条边包含起点、终点、代表关系以及权重（社会关系的亲疏）。
* 以下是构建力导向图的关键代码及相关说明：

let simulation = d3.forceSimulation() // 构建力导向图

.force('link', d3.forceLink().id(function (d, i) { return i; }).distance(function (d) { return d.value \* 100; }))

.force("charge", d3.forceManyBody())

.force("center", d3.forceCenter(width / 2, height / 2));

let link = g.append("g") // 画连接线

.attr("class", "links")

.selectAll("line")

.data(edges)

.enter().append("line")

.style('stroke-width', '1px')

.style('stroke', '#ddd');

let linkText = g.append("g") // 画连接线上面的关系文字

.attr("class", "link-text")

.selectAll("text")

.data(edges)

.enter().append("text")

.text(function (d) {

return d.relation;

})

.style("fill-opacity", 0);

let node = g.append("g") // 画圆圈和文字

.attr("class", "nodes")

.selectAll("g")

.data(nodes)

.enter().append("g")

.call(d3.drag()

.on("start", dragstarted)

.on("drag", dragged)

.on("end", dragended))

.on('dblclick', connectedNodes);;

function ticked() { // 力导向图变化函数，让力学图不断更新

nodes.forEach(function (d, i) {

d.x = d.x < 0 ? 0 : d.x;

d.x = d.x > width - 65 ? width - 65 : d.x;

d.y = d.y < 0 ? 0 : d.y;

d.y = d.y > height - 65 ? height - 65 : d.y;

})

link.attr("x1", function (d) { return d.source.x; })

.attr("y1", function (d) { return d.source.y; })

.attr("x2", function (d) { return d.target.x; })

.attr("y2", function (d) { return d.target.y; });

linkText.attr("x", function (d) { return (d.source.x + d.target.x) / 2; })

.attr("y", function (d) { return (d.source.y + d.target.y) / 2; });

node.attr("transform", function (d) { return "translate(" + d.x + "," + d.y + ")"; });

}

除了以上的实现思路和基本流程外，关于双击节点触发事件等实现都较为基础，均为调用函数等操作，在此不予赘述。

## 2.4 逻辑结构与物理结构

本程序中应用的逻辑结构为非线性图形结构，物理结构如下：

* 在可视化中，根据D3.js内力导向图的要求，我将节点存在nodes[]数组中，将边存在edges[]数组中。

var nodes = [];//其中每个元素如下

var nodes = {

"name" : null,

"loaction" : null,

"school1" : null,

"school2" : null,

"school3" :null,

"workplace" : null

}

var edges = []; //其中每个元素如下

var edge = {

"source": nodes.length - 1, //边的起点

"target": i, //边的终点

"relation": "", //边所代表的关系

"value": 3 //边的权重，即两人社会关系的远近

}

* 在寻找某一人可能认识的人（非好友）中，为了方便查找，我使用邻接表来存放社会关系网络的相应信息。由于Javascript中的数组是变长的，所以我使用二维数组Adj[][]来存放邻接表。

## 2.5 开发平台

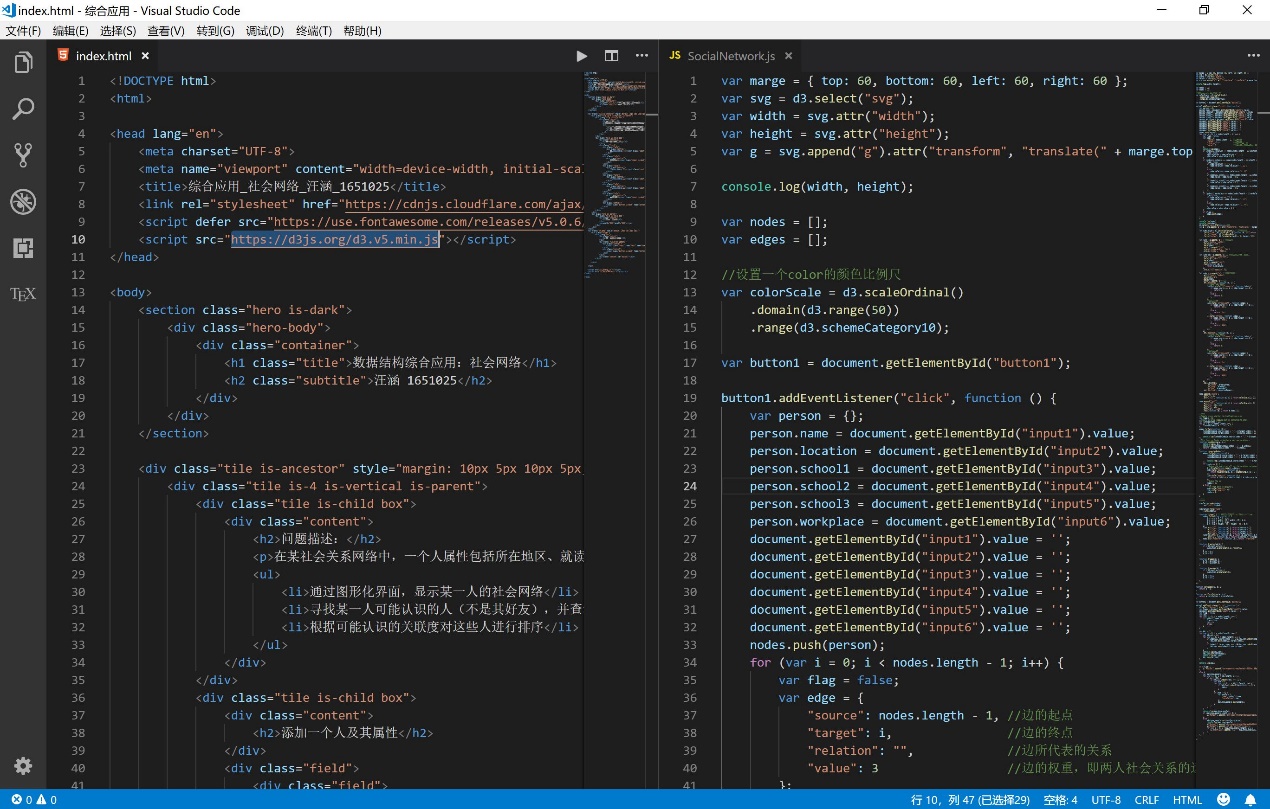
* 程序开发平台：Visual Studio Code、Chrome
* HTML5、CSS、Javascript
* JQuery(JavaScript框架、<http://jquery.com/>)
* D3.js（数据可视化Javascript库、<https://d3js.org/d3.v5.min.js>）
* Bulma(CSS框架、

<https://cdnjs.cloudflare.com/ajax/libs/bulma/0.6.2/css/bulma.min.css>)

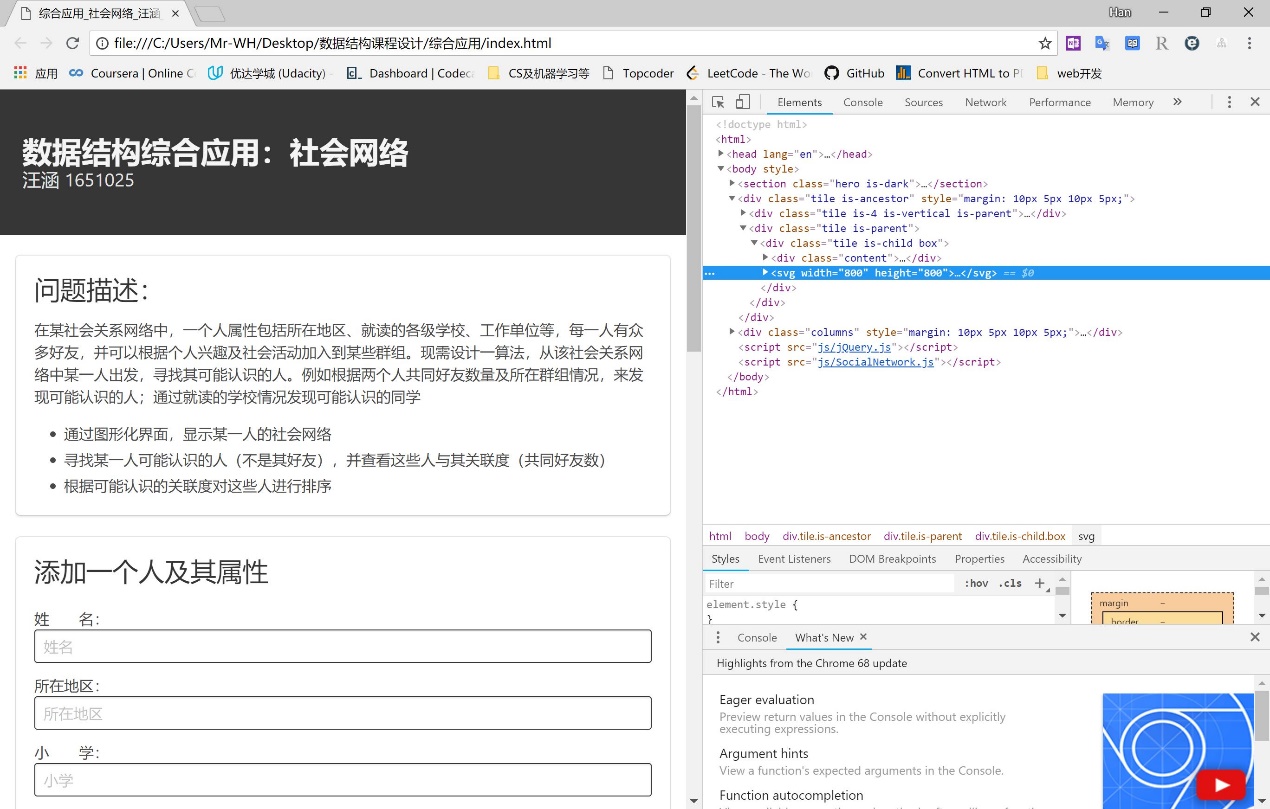
* Exe可执行文件使用Electron打包
* 运行环境：任意支持HTML5的浏览器（如Chrome、Firefox等）打开index.html或者Windows系统的电脑打开App.exe。（注意：由于在线调用了第三方框架，在运行时要保持全程联网）

## 2.6 系统的运行结果分析说明

在开发过程中使用Visual Studio Code进行开发。



在调试过程中使用Chrome开发者工具进行调试。



在开发过程中，遇到一些问题，但基本上都是在可视化方面的问题，例如界面布局，框架使用不熟等问题。没有遇到较大问题，在此举一例说明：

我发现，在每次添加新的节点时，会重复出现上一次的节点图。经过调试，我使用每次添加一个节点，都重新构建力导向图，并将上一次的替换掉，这样就解决了这个问题。

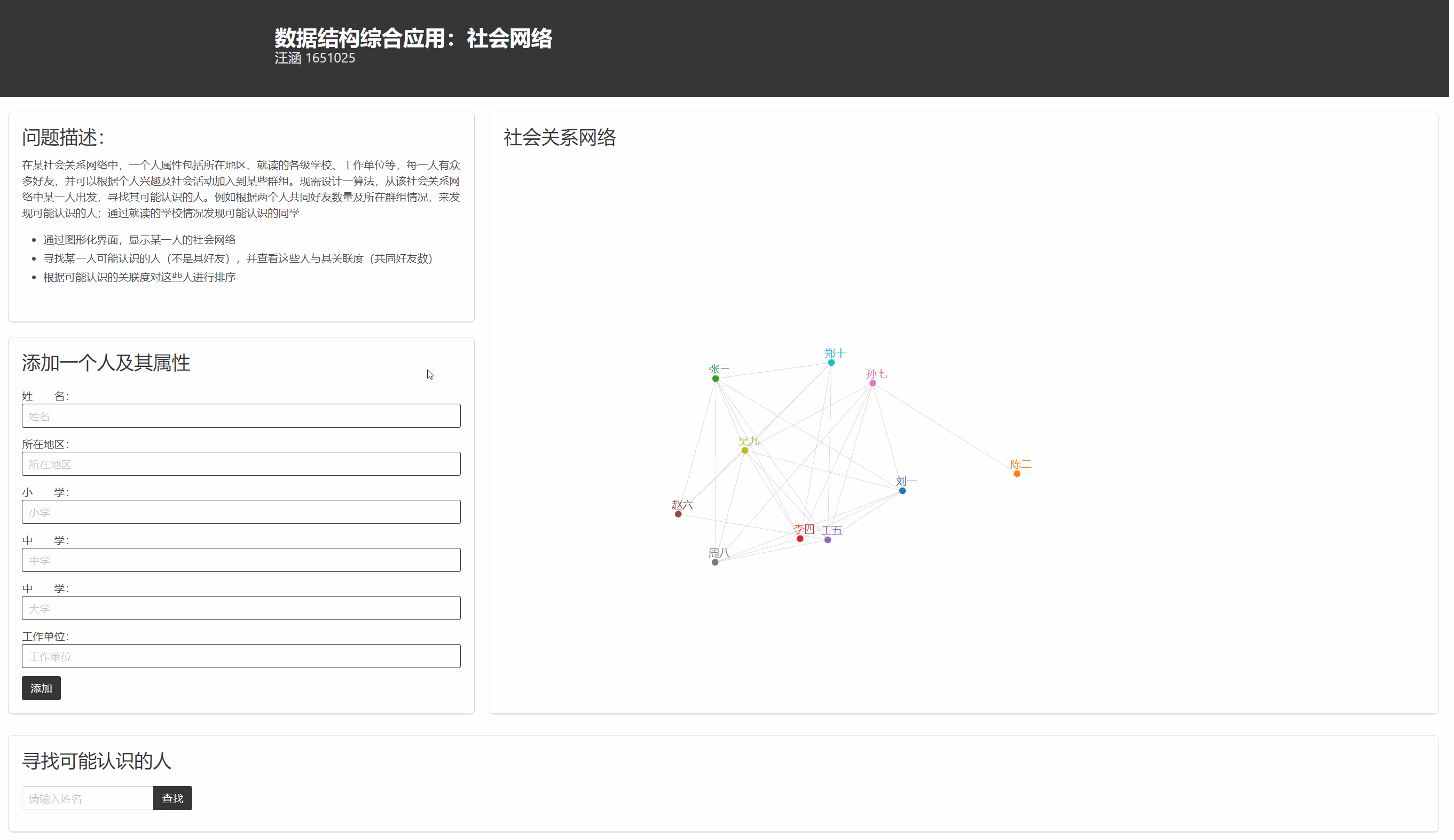
本程序运行正确、稳定，可以正常添加节点，并能正确动态显示社会关系网络的可视化，并可以与用户进行交互。可以正确查找某人可能认识的人（非好友）。

具体运行结果：

添加下列数据：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 姓名 | 所在地 | 小学 | 中学 | 大学 | 工作单位 |
| 刘一 | 深圳 | 第四小学 | 第三中学 | 清华大学 | 阿里巴巴 |
| 陈二 | 北京 | 第六小学 | 第四中学 | 同济大学 | 微软 |
| 张三 | 上海 | 第四小学 | 第一中学 | 北京大学 | 腾讯 |
| 李四 | 上海 | 第二小学 | 第三中学 | 复旦大学 | 阿里巴巴 |
| 王五 | 深圳 | 第五小学 | 第一中学 | 北京大学 | 谷歌 |
| 赵六 | 广州 | 第三小学 | 第二中学 | 北京大学 | 百度 |
| 孙七 | 北京 | 第五小学 | 第四中学 | 上海交通大学 | 阿里巴巴 |
| 周八 | 上海 | 第二小学 | 第三中学 | 上海交通大学 | 谷歌 |
| 吴九 | 广州 | 第三小学 | 第三中学 | 北京大学 | 阿里巴巴 |
| 郑十 | 广州 | 第一小学 | 第一中学 | 复旦大学 | 腾讯 |

此时社会关系网络可视化及交互效果如下（按住ctrl并单击图片可访问相应gif链接）：

[](运行效果截图等/社会网络1.gif)

输入寻找可能认识的人的人名：

如，小明:



如，刘一：



## 2.7 操作说明

* 双击打开index.html或App.exe，将会进入如下界面：



* 在输入框中添加一个人及其属性，并点击添加。





* 按照上述方式以此添加每个人及其属性。
* 用户可以将光标移动到社会关系网络中代表某人的节点上，此时显示其与他人的社会关系。用户可以双击社会关系网络中代表某人的节点上，此时高亮显示该人的社会关系网络，再次双击取消。具体方法见2.6系统的运行结果分析说明最后的gif图片。
* 在输入框中输入需要寻找可能认识的人的人名，点击查找。



# 第三部分 实践总结

## 3.1 所做的工作

本次课程设计中，我的项目均是使用HTML5+CSS+Javascript的方式来完成的。之所以本次会采用这种方式，一是因为老师没有限制大家对于程序语言和实现方式的选择，二是因为被Javascript所支持的数据可视化工具所吸引，想去学习使用，三是因为以前在做其他项目时零零散散学习过一些前端相关知识，发现自己对这个方向还是较感兴趣，希望借助此次课程设计来系统地学习一下前端的相关知识以及数据可视化的相关知识。我所做的工作包括使用CSS框架Bulma来完成界面的布局设计以及交互性，然后使用Javascript实现算法实现中二叉树的建立及统计叶子节点数，二叉树的先序、中序、后序遍历，二叉树的先序、中序线索化及遍历，可视化二叉树及动态遍历；综合应用中社会关系网络的数据结构图的建立，力导向图的建立，寻找可能认识的人（非好友）的实现。最后我再通过Electron对项目进行打包生成exe可执行文件。

## 3.2 总结与收获

通过本次课程设计我系统的学习了HTML5+CSS+Javascript等前端相关知识，并学习使用了一些如D3.js数据可视化工具，CSS框架Bulma。除此之外我也学习了一些数据可视化的知识和方法技巧。在实现数据结构的过程中，我对于数据结构的理解进一步加深，了解到数据结构在实际工作中的应用与课堂上学习的知识之间的联系和差别。在本次课程设计中，我认识到数据结构和算法的重要性，自己将来在这方面还需要继续加强。除此之外，我还锻炼了自己的编程及调试能力，提高自学能力和阅读官方文档的能力。总的来说本次课程设计实我受益匪浅。

# 第四部分 参考文献

[1] Mark Allen Weiss. *Data Structures and Algorithm Analysis in C++, Forth Edition* 电子工业出版社,2016

[2] Michael McMillan. 数据结构与算法Javascript描述 人民邮电出版社,2014

[3] Scott Murray. 数据可视化实战：使用D3设计交互式图表 人民邮电出版社,2013

[4] Nick QiZhu. D3.js数据可视化实战手册 人民邮电出版社,2014

[5] MND web docs <https://developer.mozilla.org/zh-CN/>

[6] D3.js Documentation <https://github.com/d3/d3/wiki>

[7] BULMA Documentation <https://bulma.io/documentation/>