同济大学
TONGJI UNIVERSITY

《数据结构》课程设计总结

学　　院：电子与信息工程学院

专　　业：信息安全

学　　号：1751237

姓　　名：张天然

指导老师：武妍、叶晨

2019　年　9　月　2　日

目录

第一部分　实践总结

1.1　题目

输入一组关键值，建立相应的二叉排序树，完成结点的查找和删除操作。

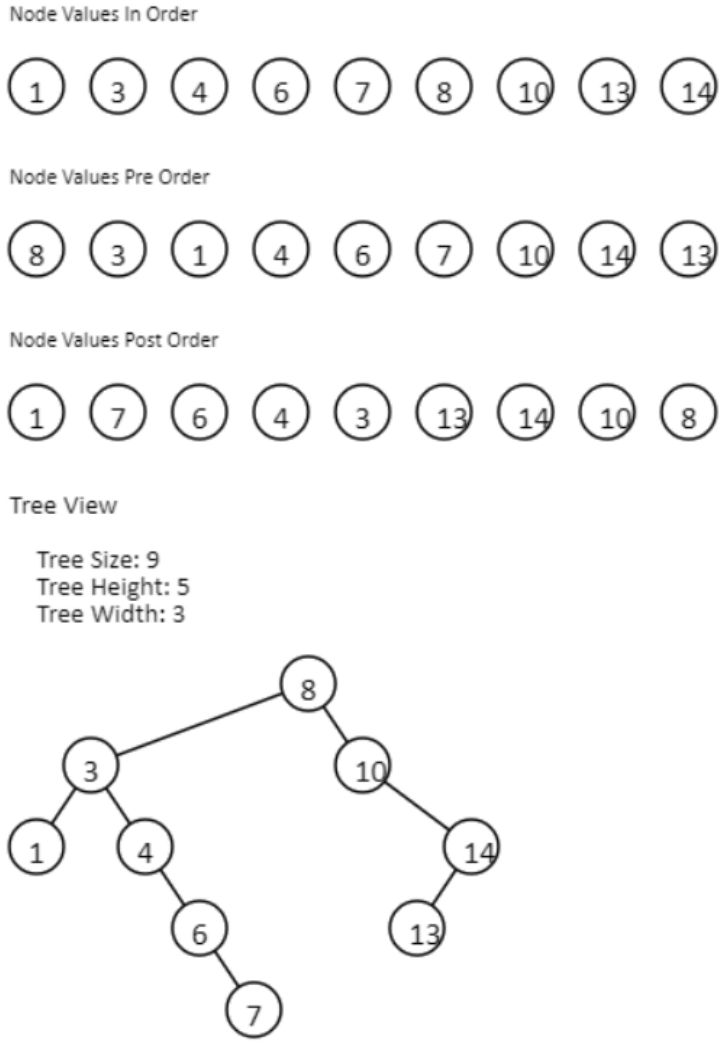
要求:

（1）　可以实现删除根节点、叶子节点以及其他任意节点的功能;

（2）　可以随时显示操作的结果。

1.2 软件功能

1.2.1 图形化实现图:



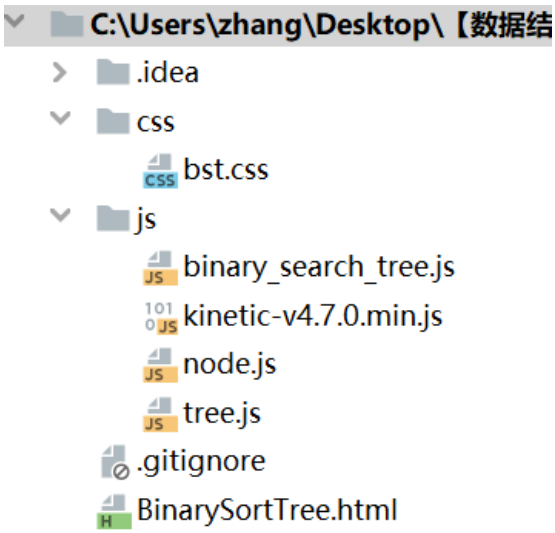
1.2.2 可实现的操作:

输入结点的值，可将其添加为排序树的结点。

（删除功能暂未实现）

1.3 设计思想

1.3.1 整体程序框架



1）由单个 BinarySortTree.html 调用各个 js,css 文件，其中 css 文件夹为 css 库的调用，其中的 css 为自己编写的排版程序，主要为了界面的美观。

2）src 文件夹为所有代码文件，叙述如下：

Node.js 为 node 类的定义；

Tree.js 为二叉树的建立和二叉树的相关信息；

BinarySortTree.js 为定义应用按钮关联的函数。

整个软件用 html 构建，在网页中显示出二叉树的具体形式。

1.3.2 bst.css

用于排版

```
body {
    margin: 10px;
    padding: 0px;
}

#description p {...}

#description a {...}

#input-div {...}

#container {...}

.kineticjs-content {...}

canvas {...}
```

1.3.3 node.js

定义 node 类:

```
// Setup BST Prey name space
BST.namespace( ns_string: 'BST.Node');

BST.Node = function(value) {...}
```

1.3.4 tree.js

```
function getHeight(node)
function getWidth(node, widthArray, level)
function getSize(node) //获取二叉树的规模参数
function addNode(node, checkNode) //给二叉树添加结点
function traverseInOrderRecursive(node, valuesInOrder) {}//中序
递归遍历
function traverseInOrderIterative(node, valuesInOrder) {
```

```
}//中序迭代遍历

function traversePreOrderRecursive(node, valuesPreOrder) //先序
递归遍历

function traversePreOrderIterative(node, valuesPreOrder) //先序
迭代遍历

function traversePostOrderRecursive(node, valuesPostOrder) //后
序递归遍历

function traversePostOrderIterative(node, valuesPostOrder) //后
续迭代遍历

function addNode(node, checkNode) //给二叉树添加结点

function inOrderLayoutNode(layer, yPos) //中序排列

function preOrderLayoutNode(layer, yPos) //先序排列

function postOrderLayoutNode(layer, yPos) //后序排列

function drawNode(layer, thisPt, radius, node) //画结点
```

核心算法：

```
162     //给二叉树添加结点
163     function addNode(node, checkNode) {
164         if (rootNode === undefined) {
165                 rootNode = node;
166         } else {
167
168             if (node.getValue() < checkNode.getValue()) {
169
170                 if (checkNode.getLeft() === undefined) {
171                     checkNode.setLeft(node);
172                 } else {
173                     addNode(node, checkNode.getLeft())
174                 }
175             } else if (node.getValue() > checkNode.getValue()) {
176
177                 if (checkNode.getRight() === undefined) {
178                     checkNode.setRight(node);
179                 } else {
180                     addNode(node, checkNode.getRight())
181                 }
182             }
183         }
184     }
```

```
308     //画结点
309     function drawNode(layer, thisPt, radius, node) {//画结点
310
311         layer.add(new Kinetic.Circle({
312             x: thisPt[0],
313             y: thisPt[1],
314             radius: radius,
315             fill: 'white',
316             stroke: 'black',
317             strokeWidth: 1
318         }));
319
320         layer.add(new Kinetic.Text({fontFamily: 'Calibri'...}));
328     }
```

1.3.5 BinarySortTree.js

报错设置和调用应用函数

```
function clearTree()//清除所有数据

function keyPress() //按钮函数
```

1.3.6 BinarySortTree.html

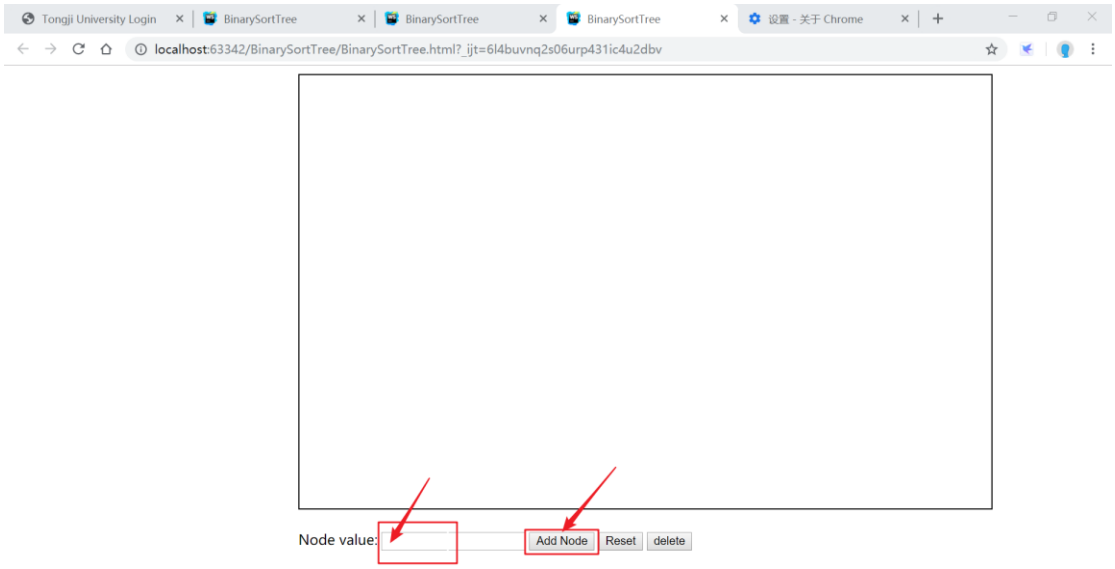
```html
<!DOCTYPE HTML>
<html>
    <head lang="en">
        <meta charset="UTF-8">
        <meta name="viewpoint" content="width=device-width, initial-scale=1">
        <title>BinarySortTree</title><!--设置网页标题-->
        <link type="text/css" rel="stylesheet" href="./css/bst.css">
        <script src="./js/binary_search_tree.js"></script><!--声明调用的脚本-->
    </head>
    <body>
        <div id="container">
        </div>
        <div id="input-div">
            <!--设置按钮及其操作-->
            Node value: <input type="text" name="value" id="value" onkeypress="keyPress();">
            <button id="add-node" type="button" onclick="addNode();">Add Node</button>
            <button type="button" onclick="clearTree()">Reset</button>
            <button type="button" onclick="deleteNode()">delete</button>
        </div>
    </body>
</html>
```

1.4 开发平台

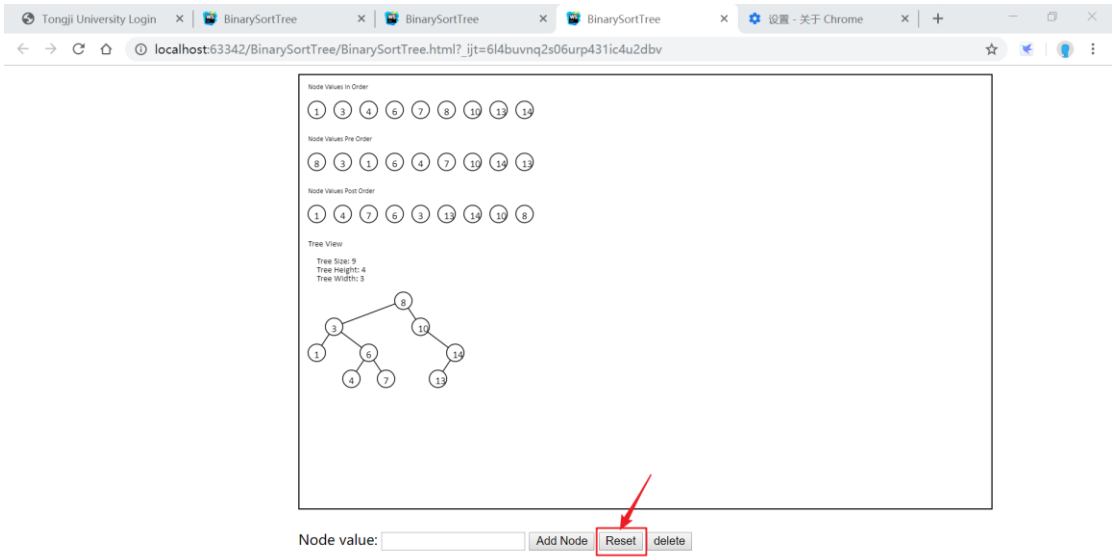
Jetbrain Webstorm 2019.2 & Google Chrome 76.0.3809.132（64 位）

网页在不同浏览器打开有不同的排版，但功能不变。

1.5 使用说明



在输入框中输入数字，点击 AddNode，即可添加改结点，框中实时显示二叉树、二叉树的规模和排序。



Reset 键可以清除已经输入的结点。

第二部分　综合应用设计说明

2.1　题目

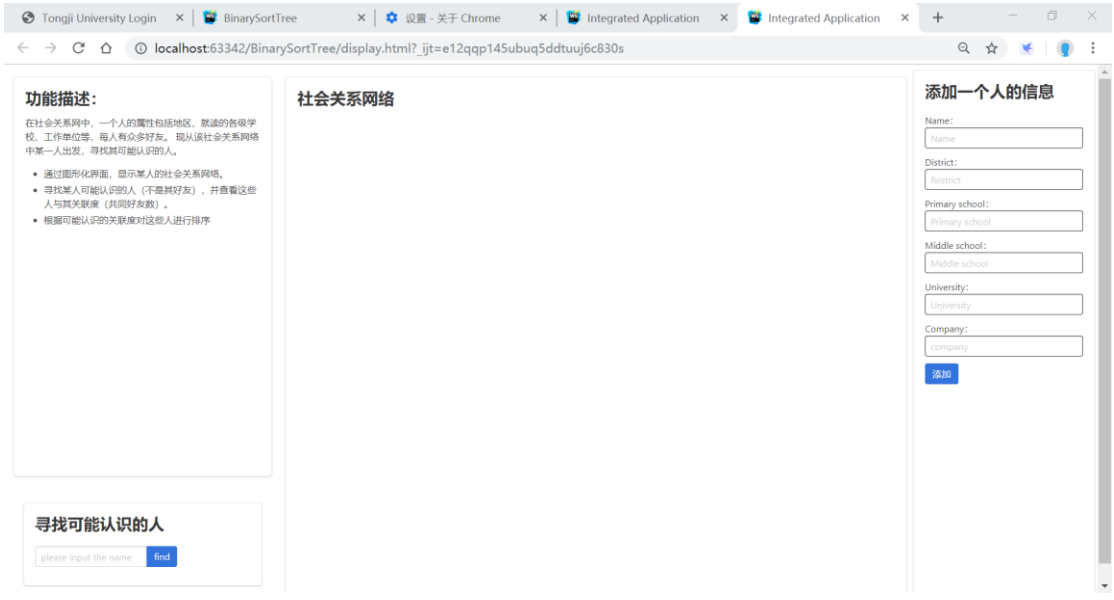
5 ★★★在某社会关系网络中，一个人属性包括所在地区、就读的各级学校、工作单位等，每一人有众多好友，并可以根据个人兴趣及社会活动加入到某些群组。现需设计一算法，从该社会关系网络中某一人出发，寻找其可能认识的人。例如根据两个人共同好友数量及所在群组情况，来发现可能认识的人；通过就读的学校情况发现可能认识的同学。

（1）通过图形化界面，显示某一人的社会网络

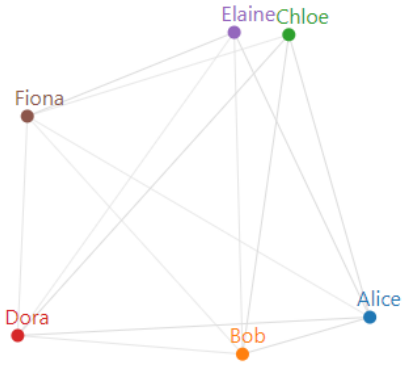
（2）寻找某一人可能认识的人（不是其好友），并查看这些人与其关联度（共同好友数）

（3）根据可能认识的关联度对这些人进行排序

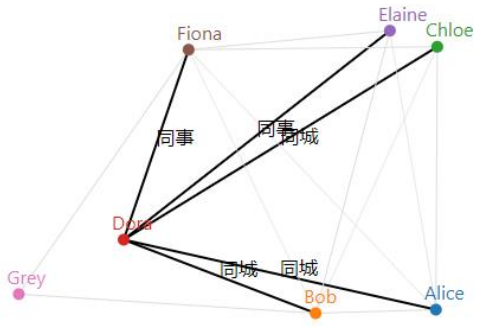
2.2　功能展示



输入个人信息后显示关系网：



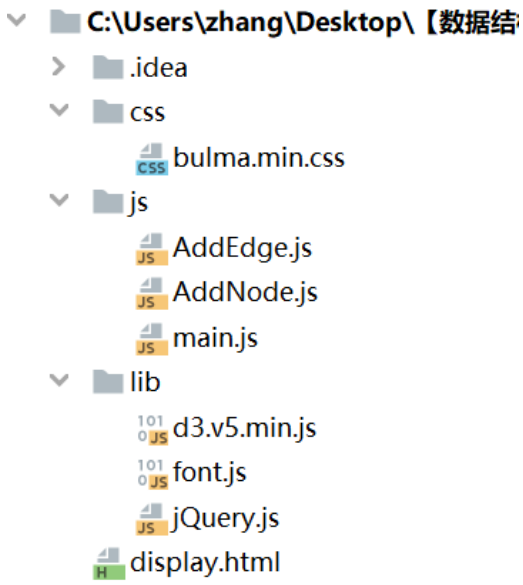
鼠标移到相应人代表的点上显示具体关系：



寻找可能认识的人：

若要寻找 A 可能认识的人，即找到和 A 可以通过至少两条直线可以连接的人。

2.3 工程组成



2.3.1 工程组成

整个文件目录由这些文件组成，其中：

bulma.min.css 为 css 所需要用到的 css 排版库文件

AddEdge.js 为增加边操作函数

AddNode.js 为增加节点操作函数

SocialNetwork.js 为 d3 库所需要调用的函数类

d3.v5 为 d3 的第五版本，具体更新时间为 2017 年 3 月

font.js 为额外调用的格式排版 js 文件

jQuery 为 jQuery 库函数文件

Display.html 为主要网页构成 HTML 文件

2.3.2 数据结构

用到的数据结构如下图所示：

```
var nodes = [];                          // store vertexs

// var node = {
//     "name": ,                         // name
//     "target": ,                       //
//     "school1": ,                      // name
//     "school2": ,                      // name
//     "school3": ,                      // name
//     "workplace":                      // name
// }
```

顶点类包含 6 个属性，分别为：

姓名、家乡、小学、中学、大学、工作单位

```
var edges = [];                          // store edges

// var edge = {
//     "source": nodes.length - 1,       // edge start
//     "target": i,                      // edge end
//     "relation": "",                   // edges relationship
//     "value": 3                        // edge value for the distance between two nodes
// };
```

边类包含 4 和属性：

分别为源点、指向点、关系字符串以及关系系数

其中关系字符串用来描述由这个点相连的两个定点的关系，而关系系数用来描述两个顶点的关系紧密度，关系系数越大，关系越紧密，算法由后面具体讲述

```
var adjTable = {};                       //Create an array recording the connection
```

用一个临界矩阵，来表示两两节点之间的相对系数关系，利用一个临界矩阵来表示两个点的关联，每个 index 表示点的序数，相应地值为边权值，即关联系数

2.4 开发平台

Jetbrain Webstorm 2019.2 & Google Chrome 76.0.3809.132（64 位）

网页在不同浏览器打开有不同的排版，但功能不变。

同时在不同版本浏览器中打开界面会有不同程度的缩放，调整大小即可。

2.5 系统的运行结果分析说明

2.5.1 SocialNetWork.js:

```
var marge = { top: 60, bottom: 60, left: 60, right: 60 };
var svg = d3.select("svg");
var width = svg.attr("width");
var height = svg.attr("height");
var g = svg.append("g").attr("transform", "translate(" + marge.top + "," + marge.left + ")");
```

此为 svg 默认参数变量设置

```
let simulation = d3.forceSimulation()              // force graph
    .force('link', d3.forceLink().id(function (d, i) { return i; })
        .distance(function (d) { return d.value * 100; }))
                                                   // the length of the line according to the similarity of node
    .force("charge", d3.forceManyBody())
    .force("center", d3.forceCenter(width / 2, height / 2));
```

与 d3_v3 不同，v4 及其以上版本选用 simulation 变量产生力导向图

```
let link = g.append("g")                                    // draw the line
    .attr("class", "links")
    .selectAll("line")
    .data(edges)
    .enter().append("line")
    .style('stroke-width', '1px')
    .style('stroke', '#ddd');
```

画边操作

```
let linkText = g.append("g")                            // draw related text on the line
    .attr("class", "link-text")
    .selectAll("text")
    .data(edges)
    .enter().append("text")
    .text(function (d) {
        return d.relation;
    })
    .style("fill-opacity", 0);
```

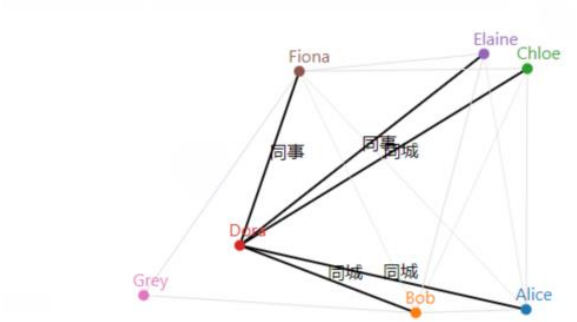
标出边上的文字操作

```
let node = g.append("g")                                    // draw circle and text
    .attr("class", "nodes")
    .selectAll("g")
    .data(nodes)
    .enter().append("g")
    .on("mouseover", function (d, i) {
```

创建点操作:

```
                                                    // display the text on the line
linkText.style("fill-opacity", function (edge) {
    if (edge.source === d || edge.target === d)
        return 1;
    else
        return 0;
});
                                                    // bold the line
link.style('stroke-width', function (edge) {
    if (edge.source === d || edge.target === d)
        return '2px';
    else
        return '1px';
}).style('stroke', function (edge) {
    if (edge.source === d || edge.target === d)
        return '#000';
    else
        return '#ddd';
});
```

函数中由调用了三个 html 设置属性函数，分别为连线文字显示设置，即当前鼠标悬停节点显示边上文字、加粗相关连线，以及设置直线颜色，如下图所示:



```
function connectedNodes() {
    edges.forEach(function (d) {
        adjTable[d.source.index + "," + d.target.index] = 1;
        adjTable[d.target.index + "," + d.source.index] = 1;

        console.log(adjTable[d.source.index + "," + d.target.index]);
    });
    if (toggle === 0) {
                                                    //Reduce the opacity of all but the neighbouring nodes
        d = d3.select(this).node().__data__;
        node.style("opacity", function (o) {
            return neighboring(d, o) | neighboring(o, d) ? 1 : 0.1;
        });
        link.style("opacity", function (o) {
            return d.index === o.source.index | d.index === o.target.index ? 1 : 0.1;
        });
        //Reduce the op
        toggle = 1;
    } else {
        //Put them back to opacity=1
        node.style("opacity", 1);
        link.style("opacity", 1);
        toggle = 0;
    }
}
```

通过临界矩阵中两两节点所对应的下标值，即关联系数的大小，返回这两个点是否链接，或者链接线的长度。其中开关 toggle 变量表示透明值是否被设置为开或关

2.5.2 AddEdge.js:

```
var flag = false;
var edge = {
    "source": nodes.length - 1, // edge start
    "target": i,                // edge end
    "relation": "",             // edges relationship
    "value": 3                  // edge value for the distance between two nodes
};
```

边的属性值如上图所示

```
if (nodes[i].location === nodes[nodes.length - 1].location) {
    flag = true;
    edge.relation = edge.relation + "同城";
    edge.value = edge.value - 0.5;
}
```

根据是否同城在 relation 字符串中是否添加“同城”，下同。

根据输入的学校情况，分别依此判断是否同学

```
if (flag)
    edges.push(edge);
```

根据是否为有关系的，即 relationship 不为空串，加入 flag 变量

2.5.3 AddNode.js:

```
function AddNode(){
    var person = {};
    person.name = document.getElementById("input1").value;
    person.location = document.getElementById("input2").value;
    person.school1 = document.getElementById("input3").value;
    person.school2 = document.getElementById("input4").value;
    person.school3 = document.getElementById("input5").value;
    person.workplace = document.getElementById("input6").value;
    document.getElementById("input1").value = '';
    document.getElementById("input2").value = '';
    document.getElementById("input3").value = '';
    document.getElementById("input4").value = '';
    document.getElementById("input5").value = '';
    document.getElementById("input6").value = '';
    nodes.push(person);
}
```

AddNode 函数很简单，由 document 得到节点 input 框中的值，从而得到 value，

添加到临时变量 person 中，之后再 push 到 nodes 数组中

2.5.4 display.html

```
<!DOCTYPE html>
<html>

<head lang="en"...>

<body>

<!--...-->

<div class="tile is-ancestor" style="..."...>


<script src="lib/jQuery.js"></script>
<script src="js/AddEdge.js"></script>
<script src="js/AddNode.js"></script>
<script src="js/main.js"></script>
</body>

</html>
```

设置框体和排版等。

第三部分　实践总结

3.1 所做的工作

自学包括 html,CSS,Javascript 的语言，明显感受到和 C++相比，这些语言在可视化方面更加便捷。

3.2 总结及感悟

原本的计划是用 C++结合 Qt 编写程序，因为大一以来一直在学习 C++，对于 python，Java，html 等则不太熟悉。但是在了解了 JavaScript 之后，发现 JavaScript 在函数编写上和 C++有很多相似之处，而 HTML 则大大简化了可视化

的操作。以前只用 html 编写过单一功能的网页，结合 JavaScript 可以实现很丰富的功能。原本觉得语言并不重要，算法才重要，现在发现不同语言在不同的领域有不同的优势，根据需求选择最合适的语言，可以事半功倍。