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：计算机科学与技术

学　　号：1651574

姓　　名：贾昊霖

指导老师：叶晨

2018　年　9　月　11　日

目录

第一部分　算法实现设计说明 1
1.1　题目 1
1.2　软件功能 1
1.2.1 图形化实现图，给用户很好的直观的体验 1
1.2.2 显示出其邻接表 2
1.2.3 输出确定起点的最短路径问题 2
1.2.4 用不同算法显示出一个图的最小生成树 2
1.3　设计思想 3
1.3.1 整体程序框架 3
1.3.2 index.html 3
1.3.3 font.css 4
1.3.4 forceGraph.js 5
1.3.5 algorithm.js 7
1.3.6 highlightLine.js 12
1.3.7 link_algorithm_graph.js 12
1.4　逻辑结构与物理结构 14
1.5　开发平台 14
1.6　系统的运行结果分析说明 16
1.6.1 调试遇到的问题一 16
1.6.2 调试遇到的问题二 16
1.7　操作说明 17
第二部分　综合应用设计说明 20
2.1　题目 20
2.2　软件功能 20
2.3　逻辑结构与物理结构 21
2.3.1 工程组成 21
2.3.2 数据结构 22
2.4　开发平台 23

注：所有代码均以 push 到我的 github 上：https://github.com/Harrypotterrrr/Data-structure-course-project

2.5　系统的运行结果分析说明 24
2.5.1 SocialNetWork.js: 24
2.5.2 AddEdge.js: 27
2.5.3 AddNode.js: 29
2.5.4 index.html 29
2.7　操作说明 32
第三部分　实践总结 34
3.1 所做的工作 34
3.2 总结及感悟 34
第四部分　参考文献 34

注：所有代码均以 push 到我的 github 上：https://github.com/Harrypotterrrr/Data-structure-course-project

第一部分　算法实现设计说明

1.1　题目

给定一个图，完成：

（1） 建立并显示出他的邻接链表

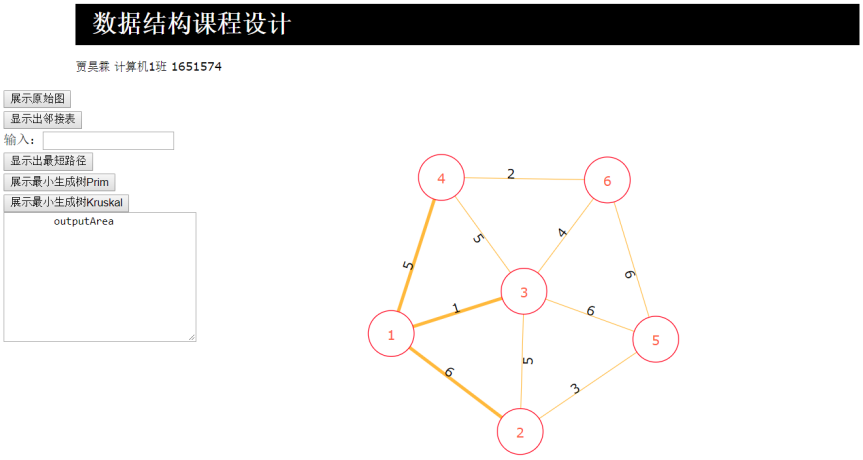
（2） 分别用 Prim 算法和 kruskal 算法构造最小生成树，随时显示其构造的过程

（3） 给出某一确定定点到所有其它定点的最短路径

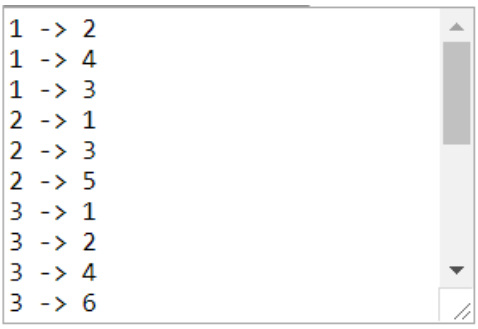
（4） 给出每一对顶点之间的最短路径

1.2　软件功能

1.2.1 图形化实现图，给用户很好的直观的体验

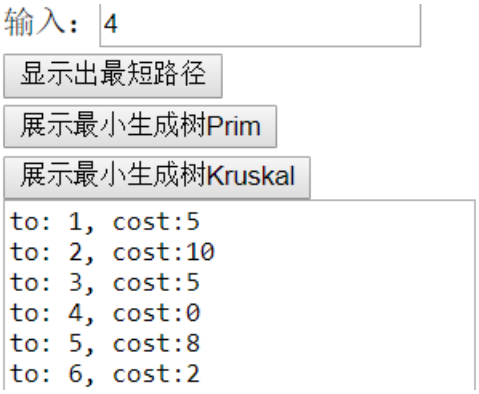


1.2.2 显示出其邻接表



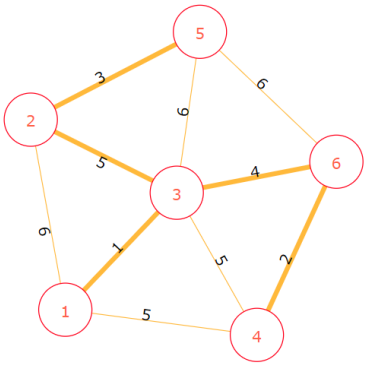
邻接表，存储方法跟树的孩子链表示法相类似，是一种顺序分配和链式分配相结合的存储结构。如这个表头结点所对应的顶点　　存在相邻顶点，则把相邻顶点依次存放于表头结点所指向的单向链表中。

1.2.3 输出确定起点的最短路径问题



在输出框中输出可以显示从一个源点到达其他节点的最短路径。

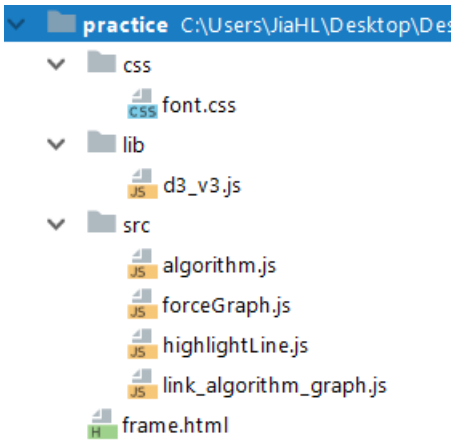
1.2.4 用不同算法显示出一个图的最小生成树



最小生成树的边由加粗的黄色边显示

1.3 设计思想

1.3.1 整体程序框架



由单个 frame.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

lib 文件夹下的 d3_v3 为大数据可视化 d3 库的库文件，下载源为 d3 官网的：

src 文件夹为所有代码文件，叙述如下：

algorithm.js 为算法主程序，里边包含了 Graph 类的实现

forceGraph.js 为 d3 库的调用即实现可视化的主要 js 程序

highlightLine 为但拿出来的函数文件

link_algorithm_graph.js 为串联 algorithm 与 forceGraph 程序的重要关联程序

整个软件用 html 构建，在网页中显示出图以及相应算法的结果，主要实现数据的输入判断、打印输出与 6 种图算法，为便于相互间配合实现子对话框，动态显示等功能。各函数相应作用如下：

1.3.2 index.html

```
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <title>数据结构课程设计</title>
    <link rel="stylesheet" href="css/font.css">
</head>
```

head 部分的 html 代码，并用 link 调用了 css 文件

如图所示为，Webstorm 下 html 种 buttons 类下的所有成员

```
<div id="buttons">
        <button onclick="GetValue()">展示原始图</button>
        <br>
        <button onclick="GetAdjList()">显示出邻接表</button>
        <br>
        输入：<input type="text" value="" id="textId">
        <br>
        <button onclick="GetDijkstra()">显示出最短路径</button>
        <br>
        <button onclick="GetPrim()">展示最小生成树Prim</button>
        <br>
        <button onclick="GetKruskal()">展示最小生成树Kruskal</button>
        <div id="para">
        <textarea id="outputArea" rows="20" cols="30">
        outputArea
        </textarea>
</div>
</div>
<div id="force"></div>
```

1.3.3 font.css

```
body {
    margin: 0 auto;
    /*font-size: 12px;*/
    font-family: Verdana;
    line-height: 150%;
}

h1 {
    font-size: 30px;
}

h2 {
    font-size: 14px;
}

h3 {
    font-size: 14px;
    font-weight: normal;
    margin-left: 100px;
}
```

自己编写的 css 文件，实现朴素的界面，简朴风格

1.3.4 forceGraph.js

```
var links = [];
var nodes = [];

var width = 800,
    height = 500;

var force, svg;
var edges_line,edges_text,circle,text;
```

此为全局变量，用于传输变量数值，使 link_algorithm_graph 文件得以调用

```
force = d3.layout.force()//layout将json格式转化为力学图可用的格式
.nodes(d3.values(nodes))//设定节点数组
.links(links)//设定连线数组
.size([width, height])//作用域的大小
.linkDistance(180)//连接线长度
.charge(-1500)//顶点的电荷数。该参数决定是排斥还是吸引，数值越小越互相排斥
.on("tick", tick)//指时间间隔，隔一段时间刷新一次画面
.start();//开始转换

svg = d3.select("#force").append("svg")
        .attr("width", width)
        .attr("height", height);
```

此为将 json 格式的文件抓换成力学图，分别设定设定节点数组，连线数组以及作用域的大小

```
edges_line = svg.selectAll(".edgepath")
    .data(force.links())
    .enter()
    .append("path")
    .attr({
        'd': function (d) {
            return 'M ' + d.source.x + ' ' + d.source.y + ' L ' + d.target.x + ' ' + d.target.y
        },
        'class': 'edgepath',
        'id': function (d, i) {
            return 'edgepath' + i;
        }
    })
    .style("stroke", function (d) {
        var lineColor;
        //根据关系的不同设置线条颜色
        lineColor = "#ffbc19";
        return lineColor;
    })
```

edges_line 变量主要为 d3.svg 种的边变量，因此很多通过这个变量可以用 attr 函数增加<edgepath>中的属性值。

```
edges_text = svg.append("g").selectAll(".edgelabel")
    .data(force.links())
    .enter()
    .append("text")
    .style("pointer-events", "none")
    //.attr("class","linetext")
    .attr({
        'class': 'edgelabel',
        'id': function (d, i) {
            return 'edgepath' + i;
        },
        'dx': 80,
        'dy': 0
    });
```

同理此为边上的文字操作，其中 dx 为可以查看的文字距离定点的距离

```
circle = svg.append("g").selectAll("circle")
    .data(force.nodes())//表示使用force.nodes数据
    .enter().append("circle")
    .style("fill", function (node) {
        var color;//圆圈背景色
        var link = links[node.index];
        if (node.name === link.source.name && link.rela === "主营产品") {
            color = "#ffffff";
        } else {
            color = "#ffffff";
        }
        return color;
    })
    .style('stroke', function (node) {
        var color;//圆圈线条的颜色
        var link = links[node.index];
        color = "#ff0000";
        return color;
```

此为增加的圈变量，用来记录图中相应的节点，其中增加了如下的方法：

```
.on("click", function (node) {
    //单击时让连接线加粗
    edges_line.style("stroke-width", function (line) {
        console.log(line);
        if (line.source.name === node.name || line.target.name === node.name) {
            return 4;
        } else {
            return 0.9;
        }
    });
    //d3.select(this).style('stroke-width',2);
})
.call(force.drag);//将当前选中的元素传到drag函数中，使顶点可以被拖动
```

即点击可以加粗线的操作，具体为 function 一个高阶函数传入一个 node 变量为系统默认的点，然后对其进行操作，即对于点击的节点而言，edges_line 变量的 style 方法种 stroke-with 又是一个高阶函数，传递进入每一个 edges_line 的 line，此 line 为所有 edges_line 中的一个，对于这个 line 进行判断，如果两个点中有一个点为点击的 node，即满足加粗操作。

```
}).attr('x', function (d) {
    // console.log(d.name+"---"+ d.name.length);
    var re_en = /[a-zA-Z]+/g;
    //如果是全英文，不换行
    if (d.name.match(re_en)) {
        d3.select(this).append('tspan')
            .attr('x', 0)
            .attr('y', 2)
            .text(function () {
                return d.name;
            });
    }
    //如果小于四个字符，不换行
    else if (d.name.length <= 4) {
        d3.select(this).append('tspan')
            .attr('x', 0)
            .attr('y', 2)
            .text(function () {
                return d.name;
```

此函数实现较为繁琐，保证了文字在旋转中可以实现夹角小于 90° 的效果，增加了程序的友好性

```
function tick() {
    circle.attr("transform", transform1);//圆圈
    text.attr("transform", transform2);//顶点文字

    edges_line.attr('d', function (d) {
        var path = 'M ' + d.source.x + ' ' + d.source.y + ' L ' + d.target.x + ' ' + d.target.y;
        return path;
    });

    edges_text.attr('transform', function (d, i) {
        if (d.target.x < d.source.x) {
            bbox = this.getBBox();
            rx = bbox.x + bbox.width / 2;
            ry = bbox.y + bbox.height / 2;
            return 'rotate(180 ' + rx + ' ' + ry + ')';
        }
        else {
            return 'rotate(0)';
        }
    });
}
```

tick 函数用于随时更新圆圈的位置，根据刷新间隔更新

1.3.5 algorithm.js

此为算法函数，最重要也是最基本的数据结构到汇聚在次 js 文件中

```
function Edge(v, w, u = undefined) {
    this.u = u;
    this.v = v;
    this.w = w;
}

function Graph(n) {
    this.inital(n);
    this.n_v = n;
    this.n_e = 0;
    this.adjList = [];
    for (let i = 1; i <= this.n_v; i++) {
        this.adjList[i] = [];
    }
}
```

定义了两个类，分别为 Edge、Graph
用于保存边的 u,v,w 以及一张图的所有信息以及求解方法

```
Graph.prototype.inital = function(n){
    this.n_v = n;
    this.n_e = 0;
    this.adjList = [];
    for (let i = 1; i <= this.n_v; i++) {
        this.adjList[i] = [];
    }
};
```

函数初始化操作，adjList 为邻接表，该表的索引为起始点，索引对应的值为指向的点

```
Graph.prototype.addEdge = function (u, v, w) {
    var edge1 = new Edge(v, w);
    var edge2 = new Edge(u, w);

    this.adjList[u].push(edge1);
    this.adjList[v].push(edge2);
    this.n_e++;
};
```

增边操作，简单易懂，在邻接表中增加边

```
var outputList = [];
Graph.prototype.showAdjList = function () {

    var text;
    for (var i = 1; i <= this.n_v; i++) {
        for (var j = 0; j < this.adjList[i].length; j++){
            text = i + " -> " + this.adjList[i][j].v;
            outputList.push(text);
            console.log(i + " -> " + this.adjList[i][j].v);
        }
    }
};
```

输出邻接表，此为题目要求 1.

```
Graph.prototype.Dijkstra = function (start) {

    let vis = [];
    let dis = [];
    let Q = [];
    for (let i = 1; i <= this.n_v; i++) {
        vis[i] = false;
        dis[i] = infinite;
    }
    dis[start] = 0;
    Q.push(new Edge(start, 0));
    while (Q.length !== 0) {
        let tmp_max = -infinite, tmp_index;
        for (let i = 0; i < Q.length; i++) {
            if (tmp_max < Q[i].w) {
                tmp_index = i;
                tmp_max = Q[i].w;
            }
        }
        var cur = Q[tmp_index];
        Q.splice(tmp_index, 1);
        for (let i = 0; i < this.adjList[cur.v].length; i++) {
            var nv = this.adjList[cur.v][i].v;
            var nw = this.adjList[cur.v][i].w;
            if (vis[nv] === false && dis[nv] > dis[cur.v] + nw) {
                dis[nv] = dis[cur.v] + nw;
                Q.push(new Edge(nv, dis[nv]));
            }
        }
    }
```

Dijkstra 最短路径求解，传入一个起始点，通过算法求解到各个点的最短路径长度，并通过 shortest_list 输出

```
var prim_list = [];
Graph.prototype.Prim = function () {

    prim_list.push(1);
    var adjVex = [];
    var lowcost = [];

    const start = 1;
    for (let i = 1; i <= this.n_v; i++) {
        lowcost[i] = infinite;
    }
    for (let i = 0; i < this.adjList[start].length; i++) {
        lowcost[this.adjList[start][i].v] = this.adjList[start][i].w;
    }
    for(let i=2;i<=graph.n_v;i++)
        adjVex[i] = start;

    lowcost[start] = 0;

    for (let i = 1; i < this.n_v; i++) {
        let tmp_min = infinite, tmp_index;
        for (let j = 1; j <= this.n_v; j++) {
            if (lowcost[j] !== 0 && tmp_min >= lowcost[j]) {
                tmp_min = lowcost[j];
                tmp_index = j;
            }
        }
```

最小生成树算法，得到最小生成树，通过 prim_list 传出相连的各个点，然后遍历求得相连的点对应的边集

```
// Union-Found
function Find(x) {
    if (father[x] === -1)
        return x;
    father[x] = Find(father[x]);
    return father[x];
}

function Union(a,b){
    var t1 = Find(a);
    var t2 = Find(b);
    if(t1 !== t2)
        father[t1] = t2;
}
```

并查集，判断是否形成环

```
Graph.prototype.Kruskal = function () {
    for (let i = 0; i <= this.n_v; i++)
        father[i] = -1;

    var edgeList = [];
    for (let i = 1; i <= this.n_v; i++) {
        for (let j = 0; j < this.adjList[i].length; j++) {
            edgeList.push(this.adjList[i][j]);
            edgeList[edgeList.length-1].u = i;
        }
    }
    edgeList.sort(function cmp(a, b) {
        if (a.w > b.w)
            return 1;
        return -1;
    });

    // console.log(edgeList);
    for (let i = 1, j = 0; i < this.n_v && j < edgeList.length; i++) {
        while(Find(edgeList[j].u) === Find(edgeList[j].v))
            j++;
        Union(edgeList[j].u,edgeList[j].v);
        var nw = edgeList[j];
        console.log("add edge",nw.u+" and "+nw.v);
        kru_list.push({from:nw.u,to:nw.v});
    }
};
```

kruskal 算法，配合并查集，先排序后得到最小生成树

```
//深度优先搜索
function dfs(v) {
    this.marked[v] = true;
    //输出一下
    if (this.adjList[v] !== undefined) {
        console.log("已访问 : " + v);
    }
    for (var i = 0; i < this.adjList[v].length; i++) {
        var w = this.adjList[v][i];
        if (!this.marked[w]) {
            this.dfs(w);
        }
    }
}
```

此为自己写的深度优先搜索

```
function bfs(s) {
    var queue = [];
    this.marked[s] = true;
    queue.push(s);
    while (queue.length > 0) {
        var v = queue.shift();
        if (v !== undefined) {
            console.log("已访问 : " + v);
        }
        for (let k in this.adjList[v]) {
            var w = this.adjList[v][k];
            if (!this.marked[w]) {
                this.marked[w] = true;
                queue.push(w);
            }
        }
    }
}
```

此为自己联系写的宽度优先搜索

1.3.6 highlightLine.js

特殊增加了一个加粗边的操作

```
function highlightLine(hl) {

    var tmp_list = hl, t;

    edges_line.style("stroke-width", function (line) {

        let one = parseInt(line.source.name);
        let another = parseInt(line.target.name);
        for (let i = 0; i < tmp_list.length; i++) {
            if (one === tmp_list[i].from && another === tmp_list[i].to)
                return 5.0;
            if (one === tmp_list[i].to && another === tmp_list[i].from)
                return 5.0;
        }
        return 0.9;
    });
}
```

时间复杂度为 O(n2)，由于时间紧迫没有考虑算法优化，走暴力了

1.3.7 link_algorithm_graph.js

链接枢纽文件

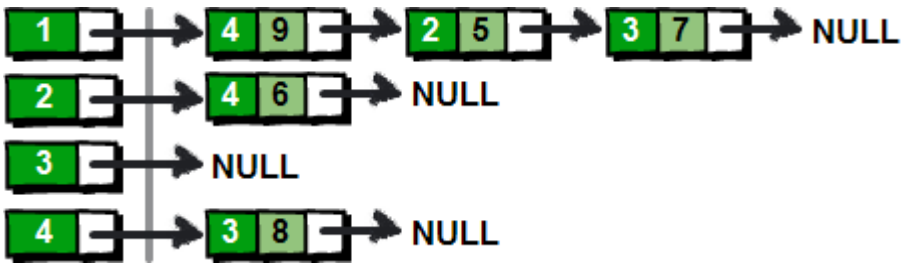
```
function GetValue() {
    // for(let i=0;i<blinks.length;i++)
    //     links.push(blinks[i]);
    d3.select('svg').remove();
    update();
}

function GetAdjList() {
    graph.showAdjList();
    var writeList = outputList.slice();
    outputList = [];
    var t = document.getElementById("outputArea");
    t.innerHTML = "";
    for (let i of writeList) {
        t.innerHTML += i + "\n";
    }
}

function GetDijkstra() {
    var text = document.getElementById("textId").value;
    if(text === "" || text === undefined || text === null){
        alert("please input a number!");
        return;
    }
    if(parseInt(text)>6 || parseInt(text)<1){
        alert("please input a number range from 1 to 6");
        return;
    }

    graph.Dijkstra(parseInt(text));
    var shor_list = shortestList.slice();
    shortestList = [];
```

```
    var t = document.getElementById("outputArea");
    t.innerHTML = "";
    for (let i of shor_list) {
        t.innerHTML += i + "\n";
    }

}

function GetPrim() {
    graph.Prim();
    var highlight_list = prim_list.slice();
    prim_list = [];
    highlightLine(highlight_list);
}

function GetKruskal() {
    graph.Kruskal();
    var highlight_list = kru_list.slice();
    kru_list = [];
    highlightLine(highlight_list);
}
```

三个函数分别为链接邻接表，链接最短路径，链接 prim，链接 kruskal 函数并得到 html 中的 id，用 document.getElementBtId()等朴素 JavaScript 语法实现（没有用 jQuery。

1.4 逻辑结构与物理结构

逻辑结构：邻接表（非邻接矩阵）



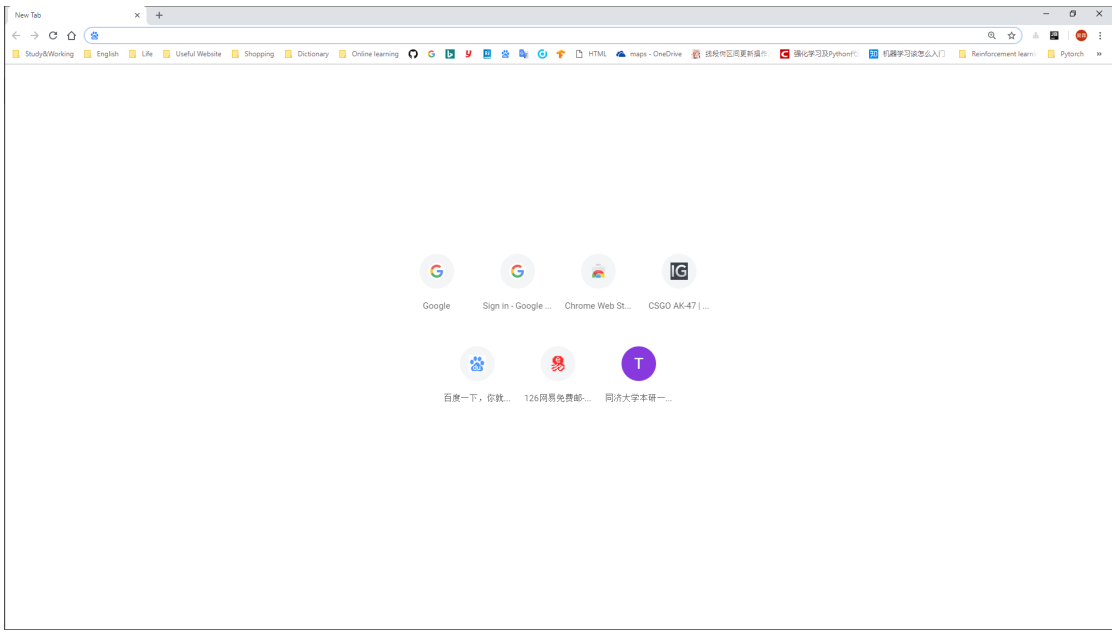
1.5 开发平台

程序开发平台：

Webstorm 2018.2

Chrome

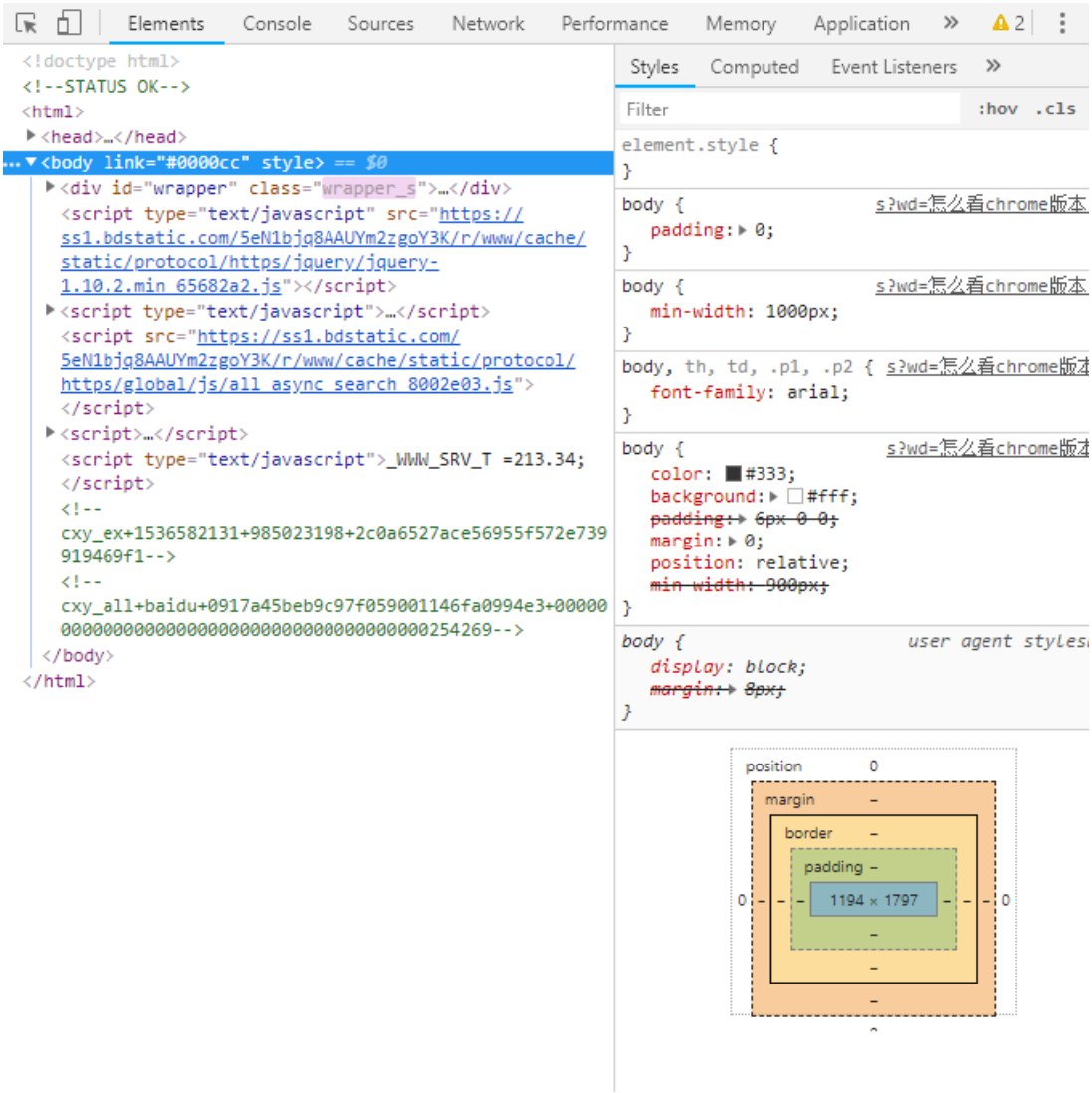
Google Chrome:	69.0.3497.81 (Official Build) (64-bit) (cohort: Stable Installs Only)
Revision:	032b3ca19e9af20182f9bd03deefc0faf4695558-refs/branch-heads/3497@{#869}
OS:	Windows
JavaScript:	V8 6.9.427.19
Flash:	30.0.0.154 C:\Users\JiaHL\AppData\Local\Google\Chrome\User Data\PepperFlash\30.0.0.154\pepflashplayer.dll
User Agent:	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.81 Safari/537.36
Command Line:	"C:\Program Files (x86)\Google\Chrome\Application\chrome.exe" --flag-switches-begin --flag-switches-end



1.6 系统的运行结果分析说明

1.6.1 调试遇到的问题一

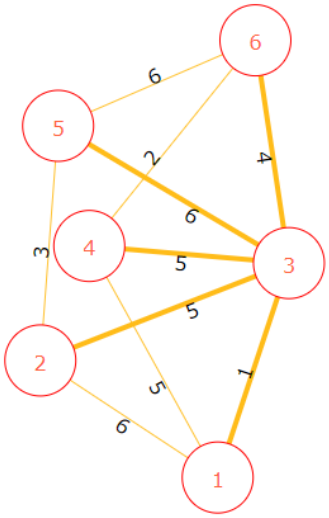
javascript 运行 html 时 Chrome 无法兼容信息，直到现在仍没解决



按网上所说，命令行中加入`--flag-switches-begin --flag-switches-end`，也无济于事..

1.6.2 调试遇到的问题二

d3 库中的图片无法显示，这个问题困扰了我 1 整天，从早上 7：30 到凌晨 1：30 浪费了我大量的时间，导致不能满足部分提议要求，促使我不得不放弃一些功能

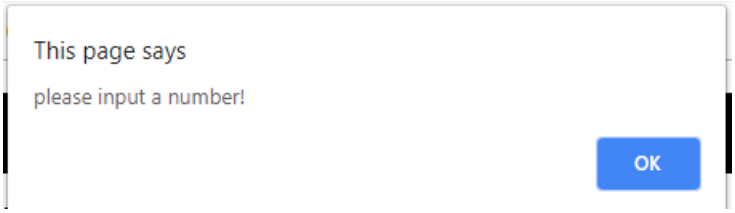


例如对动态生成最小生成树的展示等..

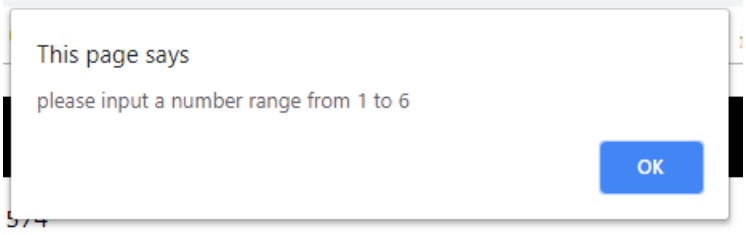
1.7 操作说明

算法实现设计软件主要由主对话框与子对话框两个部分构成。

主对话框中主要由静态文本（输入提示）、编辑框（输入区域）、五个按钮等控件，以及一个标题栏，项目栏构成。初始状态下用户通过主对话框中的编辑框随机输入一组数据后，软件判断输入数据是否合法：若输入错误，软件将警告错误类型



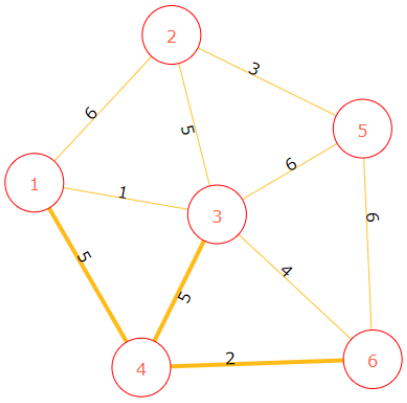
以及给定的图中判断是否输入的数字有效：

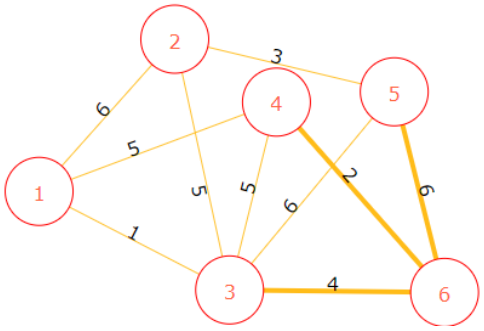


并提示用户重新输入合法数据，同时清空编辑框中输入内容，排序菜单仍保持禁止状态；若输入正确，软件将提示用户在菜单栏中选择相应排序类型，同时保存并禁止向编辑框中输入内容，禁止“确定”按钮，排序菜单则呈现使能状态。

如图所示，可以随意拉动 d3 力导向图，每个节点以及相连的边清楚地显示

如下。





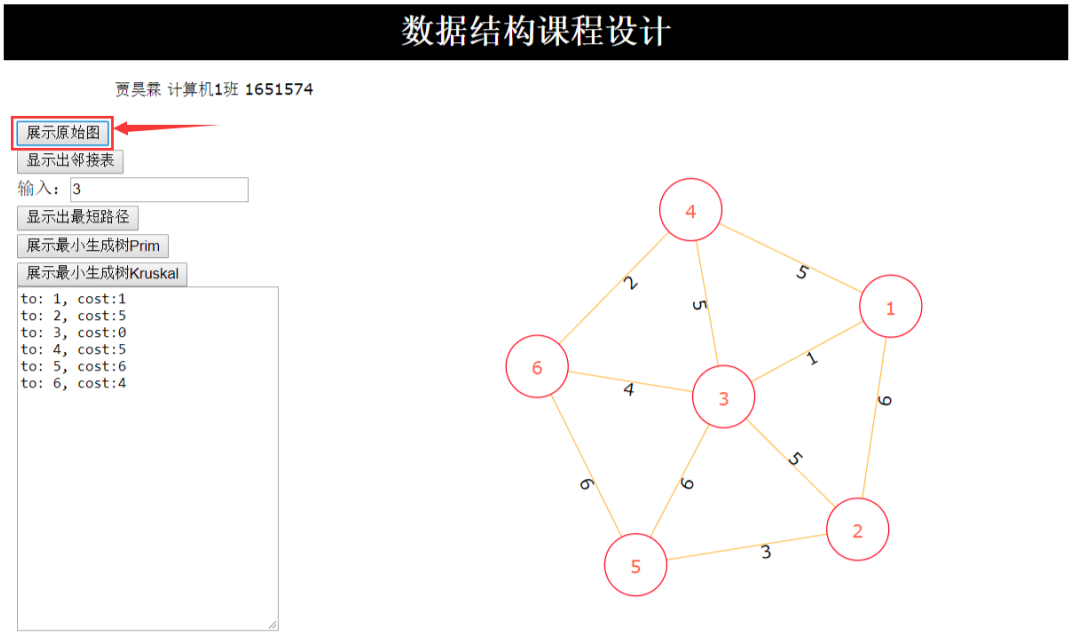
生成地最短路径输出如下：

```
to: 1, cost:1
to: 2, cost:5
to: 3, cost:0
to: 4, cost:5
to: 5, cost:6
to: 6, cost:4
```

生成地邻接表打印如下：

```
1 -> 2
1 -> 4
1 -> 3
2 -> 1
2 -> 3
2 -> 5
3 -> 1
3 -> 2
3 -> 4
3 -> 6
3 -> 5
4 -> 1
4 -> 3
4 -> 6
5 -> 2
5 -> 3
5 -> 6
6 -> 3
6 -> 4
6 -> 5
```

若点击



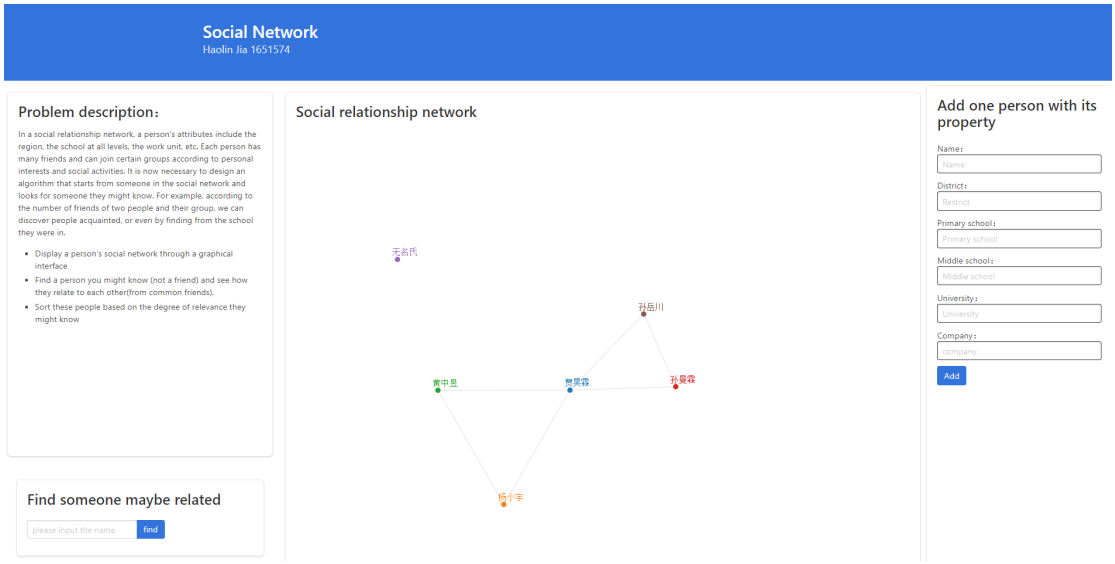
可以恢复最初地状态，即初态，重新检阅该程序

第二部分　综合应用设计说明

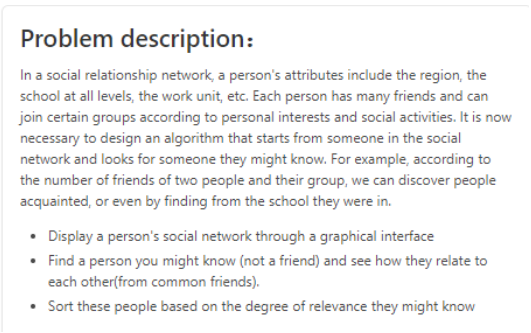
2.1　题目

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

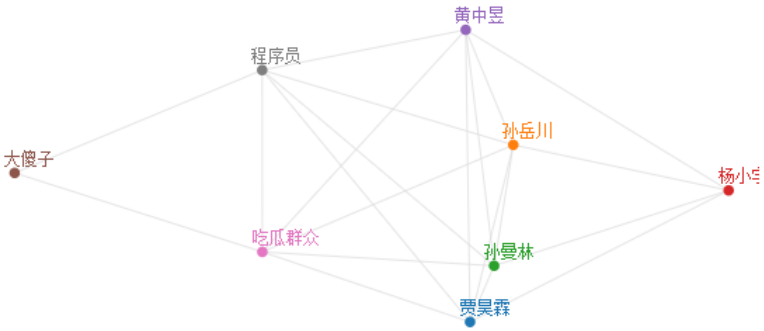
2.2　软件功能

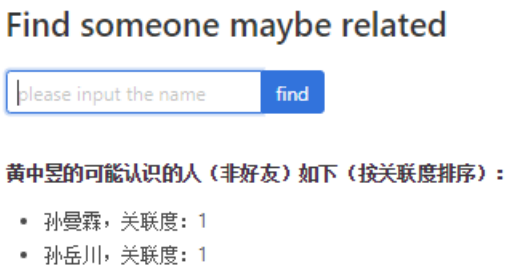


给出题目的英文提示



通过输入构建社会关系图，可视化并用相似度算法计算其相关度。

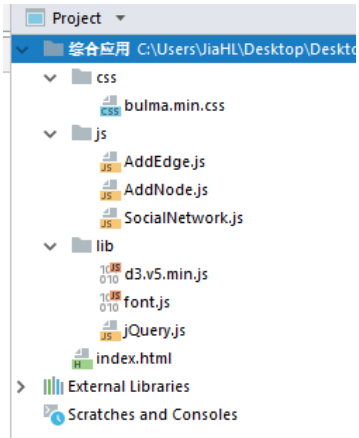




输入名称得到可能认识的人的关联度排序

2.3 逻辑结构与物理结构

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（由于前一个吸取了教训，选用了低版本导致很多函数无法实现，这个项目我选用了最新版本，并根据官方文档学习并灵活运用，构造出力导向图）

font.js 为额外调用的格式排版 js 文件

jQuery 为 jQuery 库函数文件

index.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（设计灵感来源于学长 github 开源代码！学长用的 C#编写：

（https://github.com/LiuChangFreeman/SocialRelationshipNet）

```
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
```

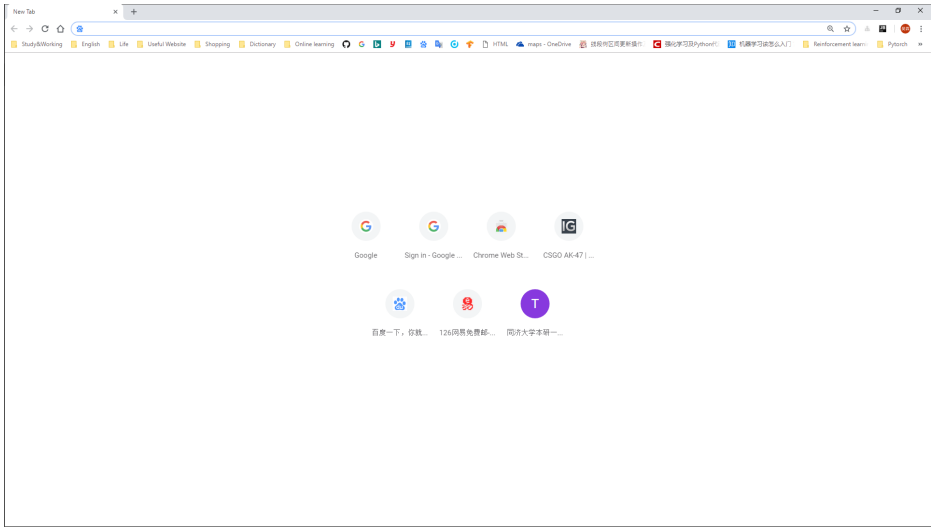
用一个临界矩阵，来表示两两节点之间的相对系数关系，利用一个临界矩阵来表示两个点的关联，每个 **index** 表示点的序数，相应地值为边权值，即关联系数

2.4 开发平台

Webstorm 2018.2

Chrome

Google Chrome:	69.0.3497.81 (Official Build) (64-bit) (cohort: Stable Installs Only)
Revision:	032b3ca19e9af20182f9bd03deefc0faf4695558-refs/branch-heads/3497@{#869}
OS:	Windows
JavaScript:	V8 6.9.427.19
Flash:	30.0.0.154 C:\Users\JiaHL\AppData\Local\Google\Chrome\User Data\PepperFlash\30.0.0.154\pepflashplayer.dll
User Agent:	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.81 Safari/537.36
Command Line:	"C:\Program Files (x86)\Google\Chrome\Application\chrome.exe" --flag-switches-begin --flag-switches-end



仍旧与前一个相同

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);
```

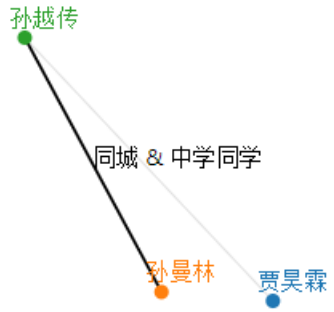
标出边上的文字操作

```
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，其中 on 函数为 javascript 常见的高阶函数，具体函数如下：

```
                                                // 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';
});
```

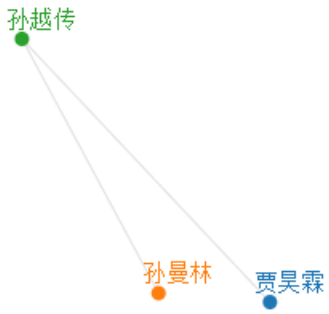
函数中由调用了三个 html 设置属性函数，分别为连线文字显示设置，即当前鼠标悬停节点显示边上文字、加粗相关连线，以及设置直线颜色，如下图所示：



而当鼠标结束悬停，离开当前节点位置时：

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

取消显示文字、取消连线加粗、取消连线颜色设置，如下图所示：




```
function neighboring(a, b) {
    console.log(a, b);
    console.log(adjTable[a.index + "," + b.index]);
    return adjTable[a.index + "," + b.index];
}
```

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

```
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;
    }
}
```

节点项链函数为关键，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

```
function ticked() {                                 // the function for updating the force graph
    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 + ")"; });
}
```

ticked 函数相同前一项目的刷新函数，故在此不过多叙述！

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
};
```

一个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”

```
if(nodes[i].school1 === nodes[nodes.length - 1].school1){
    flag = true;
    if(edge.relation === "")
        edge.relation = edge.relation + "小学同学";
    // else
    //     edge.relation = edge.relation +" & 同学";

    edge.value = edge.value-0.5;
}

if(nodes[i].school2 === nodes[nodes.length - 1].school2){
    flag = true;
    if(edge.relation === "")
        edge.relation = edge.relation + "中学同学";
    else
        edge.relation = edge.relation +" & 中学同学";

    edge.value = edge.value-0.5;
}

if(nodes[i].school3 === nodes[nodes.length - 1].school3){
    flag = true;
    if(edge.relation === "")
        edge.relation = edge.relation + "大学同学";
    else
        edge.relation = edge.relation +" & 大学同学";

    edge.value = edge.value-0.5;
}
```

根据输入的学校情况，分别依此判断是否同学

```
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;
}
```

根据输入的就职公司情况，判断两人是否为同事关系

```
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，也很暴力，利用朴素 javascript 语句，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 index.html

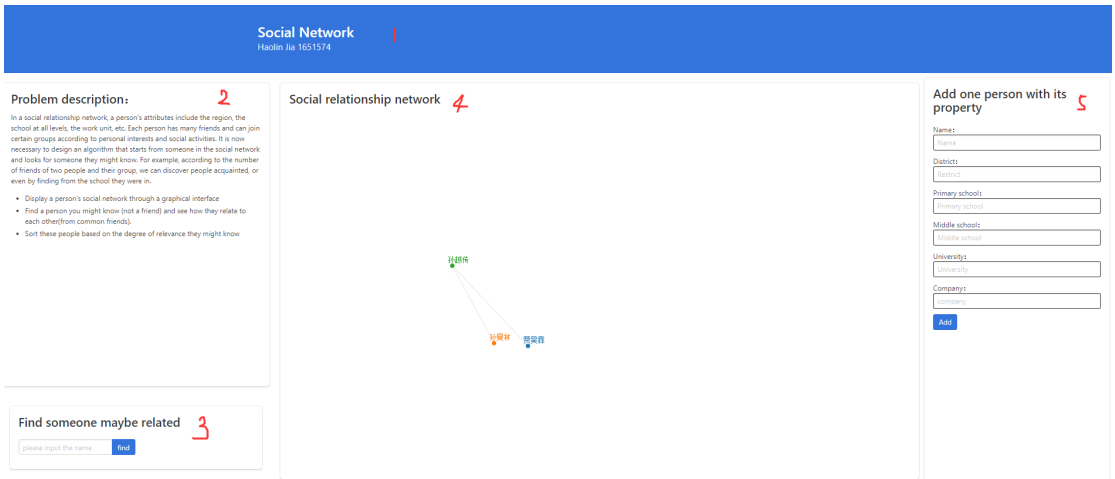
相当于 C++中的主函数（我是这样理解的）

```
<head lang="en">
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Integrated Application</title>
    <link rel="stylesheet" href="css/bulma.min.css">
    <script defer src="lib/font.js"></script>
    <script src="lib/d3.v5.min.js"></script>
</head>
```

头中加入了一些引用，如 css，jQeury，以及 d3 大数据可视化库等

```
<section class="hero is-link"...>

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

    <div class="tile is-3 is-vertical is-parent">

        <div class="tile is-child box"...>

        <div class="columns" style="..."...>

    </div>

    <div class="tile is-parent"...>

    <div class="tile is-child is-2 box"...>

</div>
```

我将代码折叠后的 html 类标签如上图所示，由 5 个模块组成，分别为头、问题描述模块、输入模块、社会关系可视化模块、以及输入模块



如上图所表示，根据学到的 class 进行定位、排版，对比上一个工程中只是简单地罗列，这个工程我运用到了 css 排版的知识，因此更加美观

```
<section class="hero is-link">
    <div class="hero-body">
        <div class="container">
            <h1 class="title ">Social Network</h1>
            <h2 class="subtitle">Haolin Jia 1651574</h2>
        </div>
    </div>
</section>
```

此为头类标签

```
<div class="tile is-child box">
    <div class="content">
        <h2>Problem description: </h2>
        <p>In a social relationship network, a person's attributes include the region, the school at all levels,
            the work unit, etc. Each person has many friends and can join certain groups according to personal
            interests and social activities. It is now necessary to design an algorithm that starts from someone
            in the social network and looks for someone they might know. For example, according to the number of
            friends of two people and their group, we can discover people acquainted, or even by finding from
            the school they were in.</p>
        <ul>
            <li>Display a person's social network through a graphical interface</li>
            <li>Find a person you might know (not a friend) and see how they relate to each other(from common
                friends).
            </li>
            <li>Sort these people based on the degree of relevance they might know</li>
        </ul>
    </div>
</div>
```

此为问题描述类标签

```
<div class="columns" style="...">
    <div class="column">
        <div class="box">
            <div class="content">
                <h2>Find someone maybe related</h2>
            </div>
            <div class="field has-addons">
                <div class="control">
                    <input class="input" type="text" name="name" id="input7"
                           placeholder="please input the name">
                </div>
                <div class="control">
                    <input class="button is-link" type="button" name="click2" value="find" id="button2">
                </div>
            </div>
            <div class="content" id="output"></div>
        </div>

    </div>

</div>
```

此为寻找相似度输入类标签

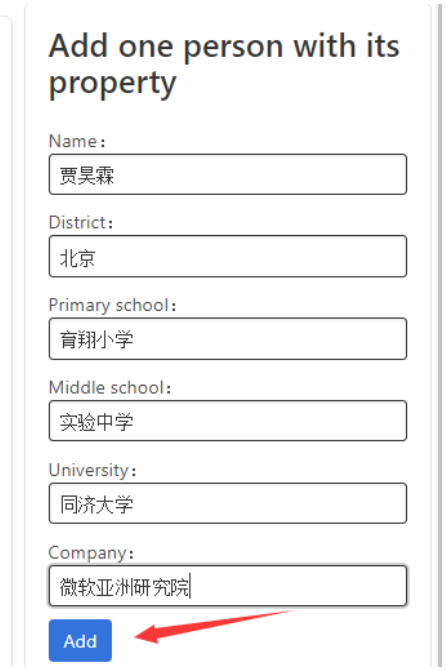
```
<div class="tile is-parent">
    <div class="tile is-child box">
        <div class="content">
            <h2>Social relationship network</h2>
        </div>
        <svg width="1000" height="800"></svg>
    </div>
</div>
```

此为大数据可视化类标签，注意此处的 svg 很好的限制了 d3 生成的力导向图的显示区域！

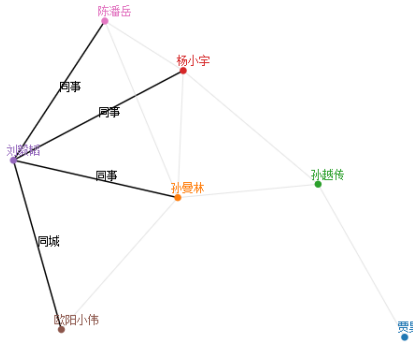
```
<div class="tile is-child is-2 box">
    <div class="content">
        <h2>Add one person with its property</h2>
    </div>
    <div class="field">
        <div class="field">
            <div class="control">
                Name: <input class="input is-dark" type="text" placeholder="Name" name="name" id="input1">
            </div>
        </div>
        <div class="field">
            <div class="control">
                District: <input class="input is-dark" type="text" placeholder="Restrict" name="location"
                                id="input2">
            </div>
        </div>
        <div class="field">
            <div class="control">
                Primary school: <input class="input is-dark" type="text" placeholder="Primary school" name="school1"
                                    id="input3">
            </div>
        </div>
        <div class="field">
            <div class="control">
                Middle school: <input class="input is-dark" type="text" placeholder="Middle school" name="school2"
                                    id="input4">
            </div>
        </div>
        <div class="field">
            <div class="control">
                University: <input class="input is-dark" type="text" placeholder="University" name="school3"
                                id="input5">
            </div>
        </div>
        <div class="field">
            <div class="control">
                Company: <input class="input is-dark" type="text" placeholder="company" name="workplace" id="input6">
            </div>
        </div>

        <div class="control">
            <input class="button is-link" type="button" name="click1" value="Add" id="button1">
        </div>
    </div>
</div>
```

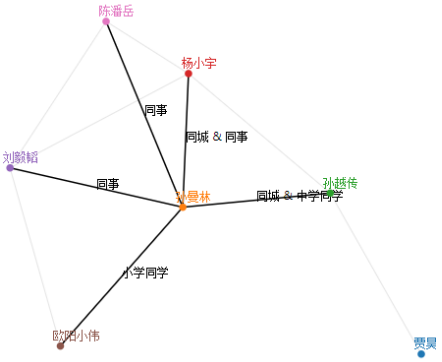
以及最后令人眼花缭乱但是非常容易实现的输入类标签组

2.7 操作说明

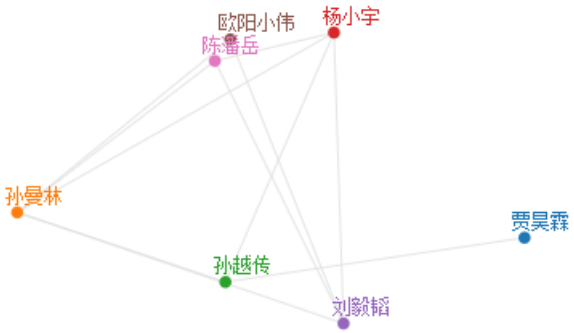


在右侧输入栏中输入相关信息，并点击 Add 按钮，即可添加一个节点的条目

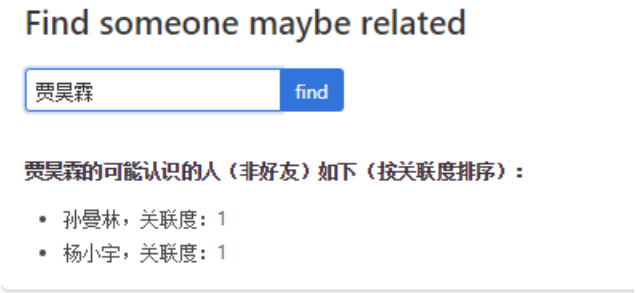




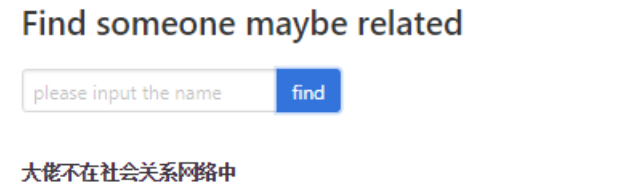
鼠标悬停在“刘毅韬”上，会显示其与周围节点的关系



也可随意拖动力导向图



输入“贾昊霖”可以返回其节点与周围节点的关系排序



在寻找框中键入“大佬”发现没有输入的人，即在节点组中不存在，会提示报错

第三部分　实践总结

3.1 所做的工作

完成所有代码的编写，以及自学包括 html,CSS,Javascript,JQuery，以及 d3 大数据可视化库的学习，还有 bulma 样式库的调用

3.2 总结及感悟

作为一个计算机科学与技术的学生，写底层、或搞科研本在我心中是第一要紧的，学习算法，运用高级的数据结构乃是所有程序员需要花大量时间的事情，然而我忽然觉得，发展方向不能太单一，不会写前端，看不懂 html，不会 javascript（网友们号称最好用的语言）毕业出去以后，也许有点丢人，因此我挤出了时间，在非常非常短的时间里学习了 javascript，html，jquery，以及 css 样式库的语法规范等。更重要的是，在一个大数据的时代，数据可视化是关键，因此我又踏上了学习 D3 的不归路... 学习过程可谓十分艰辛！调试代码也遇到了各式各样的问题，通过同学尝试解决，但是有的直到现在仍没能解决，算作这两个工程留下的遗憾吧。

不过回首这几天，我学到了真的不少东西！我倍感欣慰，熬夜，值了！！！

第四部分　参考文献

[1] 廖雪峰. javascript 教学[EB/OL]. https://www.liaoxuefeng.com/wiki/00143444 6689867b27157e896e74d51a89c25cc8b43bdb3000.

[2] 严蔚敏，吴伟民. 数据结构(C 语言版)[M]. 北京：清华大学出版社，2007.

[3] 李春葆 等. 数据结构教程(第 3 版)上机实验指导[M]. 北京：清华大学出版社，2009.

[4] 王国钧，唐国民 等. 数据结构实验教程(C 语言版)[M]. 北京：清华大学出版社，2009.

[5] cyber. D3 力导向图实现[EB/OL]. https://www.cnblogs.com/koto/p/5983693.html

[6] 菜鸟教程. HTML5 教程[EB/OL]. www.runoob.com/html/html-tutorial.html.

[7] 未知. Bulma 官方文档[EB/OL]. https://bulma.io/.

[8] 未知. D3 官方文档[EB/OL]. https://d3js.org/.

[9] 刘畅. 社会关系网络 C#[EB/OL]. https://github.com/LiuChangFreeman/SocialR

elationshipNet..

参考说明：

[1]：普及了 javascript 的语法知识

[2]: 数据结构理论与算法伪码 参考

[3]、[4]：程序中涉及相关数据结构（循环链表、循环队列、串、内部排序）的算法实现 参考

[5]：学习掌握 d3 的构建过程，以及用 svg 来绘图

[6]：学习 HTML 标签以及各个前端类极其属性

[7]：为了美化前端，选用朴素简单上手的 css 库

[8]: d3 官方文档，权威且十分有参考价值！

[9]: 参考了学长的社会关系网络开源代码，学长是用 C#写的，我吸取了一下经验，加入了我的想法实现了我的代码

注：所有代码、文档以及学习笔记均以 push 到我的 github 上：https://github.com/Harrypotterrrr/Data-structure-course-project