上 海 大 学

2019-2020 冬季学期

《数据结构（2）》课程考核报告

学 号：18120255

姓 名：姚施越

教 师：郑宇

课程考核评分表

序号	内容	分值	成绩
1	项目的设计	25	
2	项目的实现	25	
3	项目的测试	25	
4	报告的规范	25	
考核成绩			

计算机工程与科学学院

2020 年 5 月

一、注意事项：

1、考核报告必须由考生独立完成。报告提交结束后将进行查重处理，对有抄袭现象的报告，考核成绩作 0 分处理！

2、考核报告和项目代码在 6 月 1 日 12：00 之前提交到超星平台上课程作业的相关目录，逾期没有交的同学作缺考处理。

3、考核报告和项目代码分开提交（超星平台上课程作业有考核报告和项目代码二个文件夹）。考核报告用 PDF 文件格式，文件名格式为：考核报告-学号-姓名.PDF,例如：考核报告-18120000-张三.PDF。二个项目的代码分别放在二个文件夹（P1 和 P2）中，一起压缩打包，文件名格式为：项目代码-学号-姓名.RAR,例如：项目代码-18120000-张三.RAR。

二、考核题目

题目 1：远离新冠病毒

[问题描述]

近年来一种新型冠状病毒悄悄地在人类和动物之间潜伏、传播，2020 年初突然爆发，很多人被感染，甚至失去了宝贵的生命。面对突如其来的病毒，中国政府果断决策，进行了居家隔离，甚至封城处理。经过几个月的艰苦奋战，病毒的传播得到了初步的控制，人们为了生活、工作和学习开始了有限制的流动。由于外面的疫情还没有完全消失，所以人们外出时必须尽可能选择安全系数高的道路出行。

现在输入一个地图信息，包括 n 个区域和 m 条道路（n<=100，m<=1000），每段道路有一个安全值（0<=安全值<=150），安全值越大的道路越安全。所以当你从某点去往另一点时，总希望所经过路段的安全值的最小值最大。现在输入一些询问，每次询问输入 2 个点，输出这两个点之间所有路段最小安全值最大的路径。如图 1 所示，顶点 2 和 7 之间的最小安全值最大的路径为：2 4 6 7，该路径上路段最小安全值为 80。

[输入数据]

输入文件中第一行有二个整数：n、m。n 表示图中的顶点数（在无向图中，我们将所有结点从 1 到 n 进行编号），m 表示图中的边数；接下来 m 行，每行用三个整数 a,b,c 描述一条连接结点 a 和 b 的边，以及 ab 路段上的安全值 c；再接下来是若干行的询问，每个询问输入两个整数 a 和 b，表示要询问的两个顶点，输入二个-1 表示询问结束。

[输出数据]

输出结果有若干行，每行表示一次询问的结果，如果 a 和 b 是连通的则输出所选择路径的最小安全值和具体路径（最小安全值与路径之间用冒号分隔，顶点编号之间用空格分隔），否则输出“no path”。

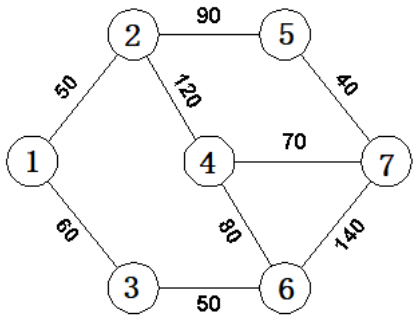

图 1 交通网络图

[输入样例 1]

7 9 3

1 2 50

1 3 60

2 4 120

2 5 90

3 6 50

4 6 80

4 7 70

5 7 40

6 7 140

2 7

-1 -1

[输出样例 1]

80：2 4 6 7

[输入样例 2]

7 6

1 2 80

1 3 60

2 4 120

3 6 70

4 6 50

5 7 40

2 6

1 7

-1 -1

[输出样例 2]

60：2 1 3 6

no path

[加分内容]

如果两个点之间有多条路径的最小安全值的最大值相等，则输出其中路径上道路条数最少的一条。如果道路条数也一样，则输出终点的前驱结点序号相对小的路径。

题目 2：例句搜索

[问题描述]

在日常使用英文的时经常会遇到一些生词，此时人们一般都会借助词典或词典类工具，查找这个单词的词义、词性等。但是，如果仅仅知道词义和词性还不是太容易掌握其地道的用法，很可能出现中国式的英语的现象。所以，直接借鉴别人写的例句是一种比较好的方法。本题目要求做一个实用的根据单词找相应的例句的搜索程序。

[基本要求]

输入一个英文单词，返回相应的英语例句。具体步骤：

1、准备语料：寻找一些英文文章，例如托福、GRE 文章、英文小说、英语新闻等。

2、处理语料：对准备好的语料进行清理、分词、建立索引、生成词典等。

3、根据索引进行查询：输入单词，返回相应的例句，要求支持重复查询。

[加分功能]

1、支持解释单词的意思。中英文解释都可以，需要自己建词库。

2、支持不断增加语料库。

3、支持一些相关信息查询。如同义词、反义词、词形变化、固定搭配等。

4、其它功能，比如例句好坏评价，语法分析之类。

[参考网站]

http://www.jukuu.com/index.php

本题单词检索部分的功能与该网站类似，其搜索结果的右侧有许多可以改进的功能。

三、考核要求

1、编写源程序

根据题目要求，采用模块化程序设计方法进行程序设计，要求程序结构清晰。上述各个功能模块要求分别用函数实现，在主函数中设计一个简易菜单，通过调用这些函数完成题目要求的功能。代码书写要规范，有简要的注释，给出函数说明。

2、撰写课程考核报告

考核报告内容包括：每个题目的主要数据结构和算法设计、测试过程以及课程总结等。

《数据结构（2）》课程考核报告

一、题目 1：远离新冠病毒

1．主要数据结构

对于本题目，我采用 Html + JavaScript 语言进行编写，接下来罗列所使用的数据结构。

1.1 GraphMat 类（图类）

（1）顶点

本题是一道典型的图论问题，首先要做的自然是图类的定义，其中要考虑到顶点和边的表示，顶点的表示比较容易，采用数组来保存，在图类里，可以通过其在数组中的位置来引用。

（2）边

再考虑边，它们描述了图的结构，图的实际信息都保存在边上面，在本题中，我采用邻接表以及邻接矩阵两种形式存储，用于后续功能的实现。

首先是邻接表，由顶点的相邻顶点列表构成的数组，并以此顶点作为索引。使用此种方案，在程序中先引用一个顶点，可以访问与这个顶点相邻的所有顶点的列表。具体结构参见示意图 1-1。

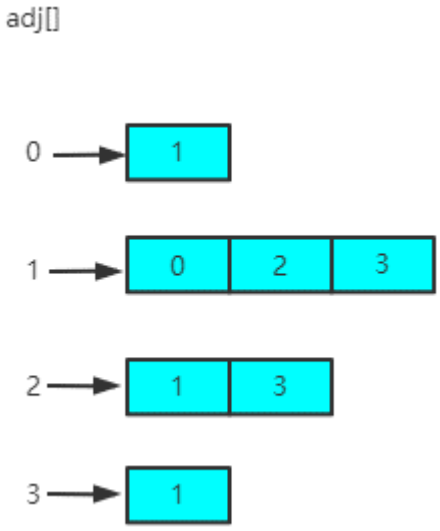

图 1-1 以邻接表表示的顶点数为 4 的图

其次是邻接矩阵，它是一个二维数组，其中的元素表示两个顶点之间是否有一条边。初始时，将图类中的 this.data 对角元素设为 0，其余元素设为无穷大。

1.2 Stack 类（栈类）

该类下定义了栈的特殊元素及常见操作，具体参见示意图 1-2。在这里事先定义此类，在后续实现寻找路径这一需求时会用到。

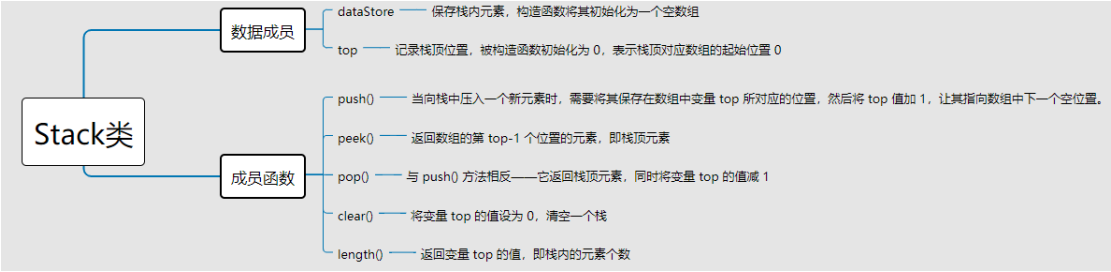

图 1-2 Stack 类的元素及操作

1.3findAllPaths()函数（使用双栈）

这个函数用于寻找两点之间的所有路径，选用的数据结构为栈，使用非递归，即循环结合双栈的方法实现。栈是一种高效的数据结构，因为数据只能在栈顶添加或删除，所以这样的操作很快，而且容易实现。双栈分别为主栈和副栈：

主栈：元素为单个节点，用于存放当前路径上的节点，最后从下至上输出。

副栈：元素为主栈对应元素的邻接表；该栈用于辅助主栈，长度也和主栈一致。

这里没有选择递归算法的深度优先遍历（DFS），因其可能会让栈溢出；而栈的结构也与使用队列的广度优先遍历（BFS）类似。本算法利用带权无向图的邻接矩阵存储结构，通过循环加双栈的思路来寻找两点间的所有路径。算法本身并不复杂，效率较高。

具体的实现将在下文的算法设计中的 2.2.3 中展开，在此不再赘述。

2. 主要算法设计

2.1 系统功能模块简述

主要功能模块分为“文件读入”、“输出邻接矩阵”、“输出安全道路”、“绘制节点图例”、“控制台查看输出”。其功能流程图如图 2-1 所示。

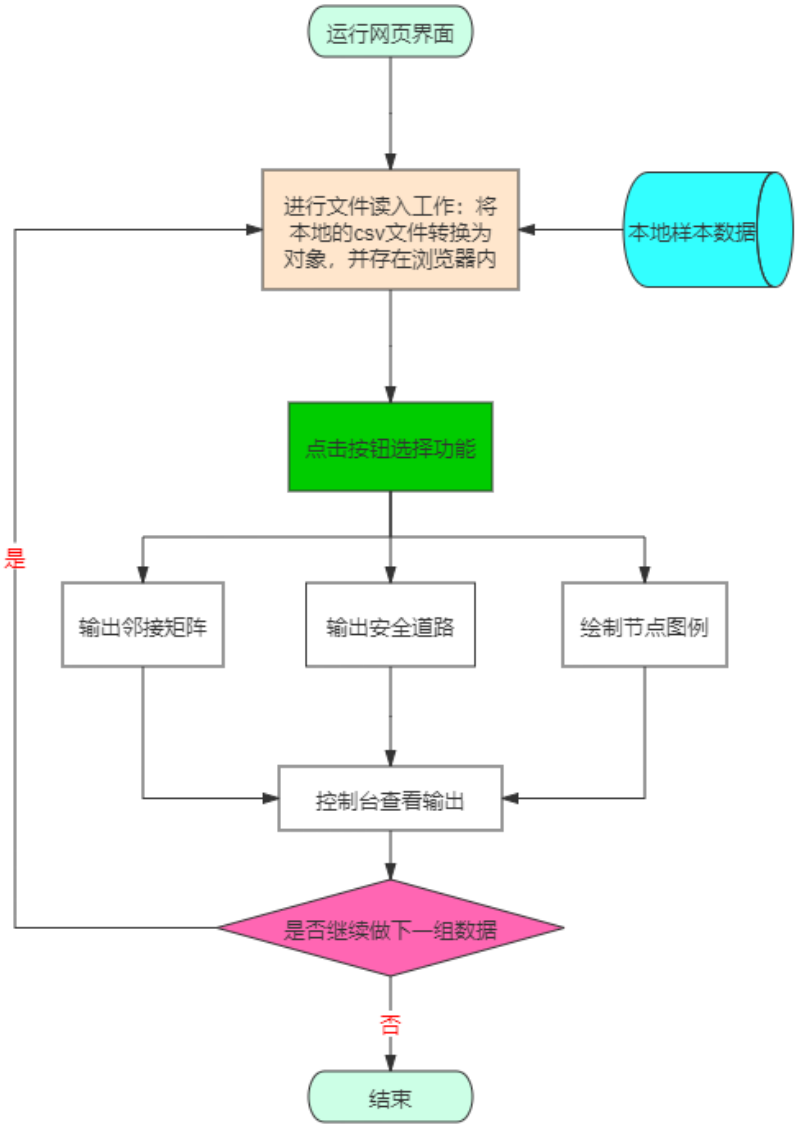

图 2-1 系统功能流程图

2.2 具体实现思路

接下来，将会对程序的各个模块的算法设计逐一进行阐述。

2.2.1 文件读入

文件读入的功能是将本地的 csv 文件转换为对象，并存在浏览器内。其中，读入文件的部分在 csv2arr.js 文件中，使用到了 Web API 中的 FileReader 对象，它允许 Web 应用程序异步读取存储在用户计算机上的文件（或原始数据缓冲区）的内容，使用 File 或 Blob 对象指定要读取的文件或数据。其语法遵循 jQuery 库（一个 JavaScript 函数库），对于 DOM 操作有着大大的简化。其最核心的属性、函数设计如下。

```
var fReader = new FileReader();
fReader.readAsDataURL( $(this)[0].files[0] );//开始读取指定的 Blob 中的内容。一旦完成，//result 属性中将包含一个 data: URL 格式的 Base64 字符串以表示所读取文件的内容。
$fileDOM = $(this);
fReader.onload = function(evt){
    var data = evt.target.result;
//  console.log( data );
    var encoding = checkEncoding( data );
//  console.log(encoding);
    //转换成二维数组，需要引入Papaparse.js
    Papa.parse( $($fileDOM)[0].files[0], {
        encoding: encoding,
        complete: function(results) {
//          console.log(results);
            var res = results.data;
            if( res[ res.length-1 ] == ""){
                //去除最后的空行
                res.pop();
            }
            callback && callback( res );
        }
    });
}
```

对于读取到的文件内容，采用 Web API 中的 localStorage 属性，其允许访问一个 Document 源的对象 Storage；存储的数据将保存在浏览器会话中。存储在 localStorage 的数据可以长期保留；当页面被关闭时，存储在 localStorage 的数据不会被清除。

```
//1.利用localStorage存在浏览器
var storage = window.localStorage;
// localStorage 中的键值对总是以字符串的形式存储，故需要使用stringify函数转为字符串
myarrString = JSON.stringify(myarr);
//增加一个数据项目名为myarr
storage.setItem("myarr", myarrString);
```

```
//2.利用 localStorage 读取存在浏览器的数据
var storage = window.localStorage;
var json = storage["myarr"];
//storage.clear();//移除所有的 localStorage 项
myarr = JSON.parse(json);//解析 JSON 字符串，构造由字符串描述的 JavaScript 值或对象
console.log(myarr);
console.log(typeof myarr);//输出 Object，即其类型为对象
```

2.2.2 输出邻接矩阵

输出邻接矩阵的功能在 index 2.0.html 文件中，定义 myshowGraph 函数。首先对于此前已转化为对象的 myarr 进行观察，第 0 行为顶点数和边数，而在第 1 行后，每一行有三个数据分别为起点、终点、权值，因此对这些含有三个数据的行，进行加边操作。其函数设计如下。

```
function myshowGraph(myarr){
    for (var i = 1; i < myarr.length; i++){
        // 第 0 行为顶点数和边数，从第 1 行开始
        // 接下来进行加边操作以及找路径操作
        var len = myarr[i].length
        var a = myarr[i][0]
        var b = myarr[i][1]
        var c = myarr[i][2]
        if (myarr[i][2] > 0){
            graph.addEdge(a, b, c)
        }
        else{
            if (a > 0 & b > 0){
                mylinjiejuzhen = graph.showGraph()
                console.log("下为邻接矩阵：")
                console.log(mylinjiejuzhen)
            }
        }
    }
}
```

对于有着三个数据的行，按照起点、终点、权值这一形式加入到邻接矩阵中，在 MyGraphMat.js 文件中，这一函数命名为 addEdge(v1, v2, w)。其函数设计如下。

```
function addEdge(v1, v2, w) {
    // 当调用这个函数并传入顶点 A 和 B 以及权值时，
    // 函数会将 this.data 的(v1,v2)和(v2,v1)的值均设为 w，最后将边数加 1
```

```
        this.data[v1][v2] = w;
        this.data[v2][v1] = w;
        this.edges++;
}
```

至于 showGraph 函数，在 MyGraphMat.js 文件中，作为 GraphMat 类的成员函数，利用了 push 方法将 this.data 元素添加到数组的末尾，并做一些处理，使得显示更直观。其函数设计如下。

```
function showGraph() {
        for (var i = 1; i <= this.vertices; ++i) {
                this.linjiejuzhen[i] = new Array();
                this.linjiejuzhen[i].push(i + ":");
                for (var j = 1; j <= this.vertices; ++j ) {
                        if (j <= this.vertices){
                                this.linjiejuzhen[i].push(this.data[i][j] + ' ');
                        }
                        if (j === this.vertices){
                                this.linjiejuzhen[i].push('; ');
                        }
                }
        }
        return this.linjiejuzhen
}
```

2.2.3 输出安全道路

输出安全道路的功能在 index 2.0.html 文件中，定义 myfindPaths 函数。首先对于此前已转化为对象的 myarr 进行观察，与输出邻接矩阵同理，第 0 行为顶点数和边数，而在第 1 行后，每一行有三个数据分别为起点、终点、权值，因此对这些含有三个数据的行，进行加边操作。而对于第 1 行之后的含有两个数据的行，若为两个顶点，则寻找二者之间的路径；若均为-1，则退出。最后，对于有路径的情况，进行寻找安全道路的操作；否则，输出 no path。其函数设计如下。

```
function myfindPaths(myarr){
        for (var i = 1; i < myarr.length; i++){
                // 第 0 行为顶点数和边数，从第 1 行开始
                // 接下来进行加边操作以及找路径操作
                var len = myarr[i].length
                var b = myarr[i][1]
                var c = myarr[i][2]
                if (c > 0){
                        graph.addEdge(a, b, c)
                }
```

```
        else{
            if(a == -1 & b == -1){
                console.log("询问结束！")
            }
            else if (a > 0 & b > 0){
                console.log("开始询问！")
                console.log("from"+a+"to"+b)
                // var mypath1 = new Array()
                mypath1 = graph.findAllPaths(a,b) //寻找两点之间所有路径
                if (mypath1 === null){
                    return
                }
                else{
                    mylinjiejuzhen = graph.showGraph()
                    //寻找安全道路
                    mypathweight = graph.findMinPath(mypath1,mylinjiejuzhen)
                }
            }
        }
    }
}
```

对于 findAllPaths 函数，在 MyGraphMat.js 文件中，作为 GraphMat 类的成员函数，利用了循环结合双栈的思想。使用到的栈是之前定义的 Stack 类。

下面我将以下图为例，具体解释如何获取两点之间路径的过程。

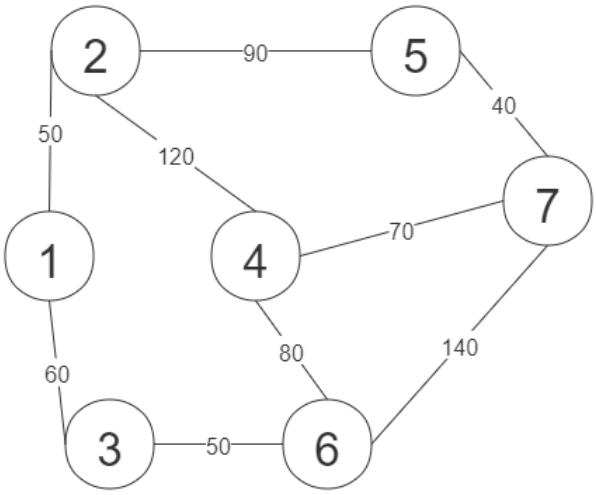

图 2-2 交通网络图（示例）

节点 2 到节点 7 的路径共有 5 条：

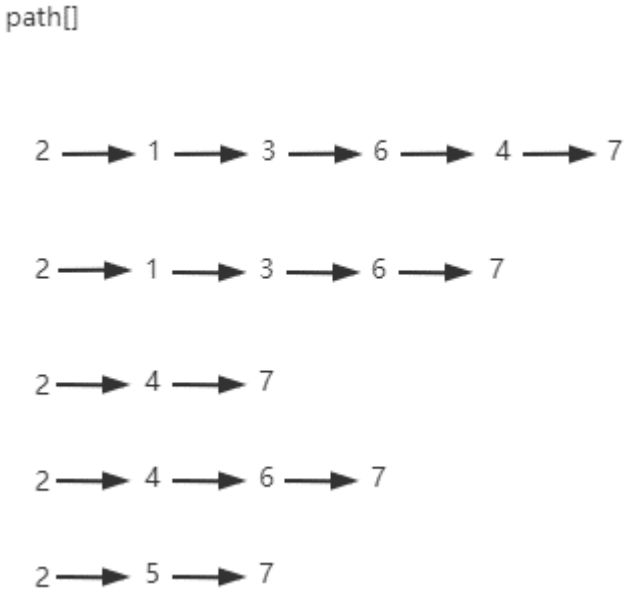

图 2-3 节点 2 到节点 7 的所有路径

（1）首先，初始化两个栈，分别为主栈和副栈。

主栈：元素为单个节点，用于存放当前路径上的节点，最后从下至上输出。

副栈：元素为主栈对应元素的邻接表；该栈用于辅助主栈，长度也和主栈一致。

（2）第一步，入栈。

将节点 2 压入主栈，同时将节点 2 的邻接表[1，4，5]压入副栈。

图 2-4 第一步入栈结果

（3）第二步，入栈。

取出副栈栈顶的第一个元素 1，将其压入主栈；同时将剩下的邻接表[4，5]压回副栈。

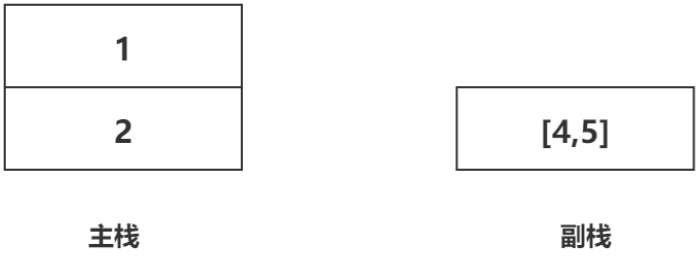

图 2-5 第二步入栈结果 1

接下来，查看主栈栈顶元素 1，其邻接表为[2，3]，节点 2 已经在主栈中，故将其剔除后的[3]压入副栈。后续不断进行这一入栈操作。

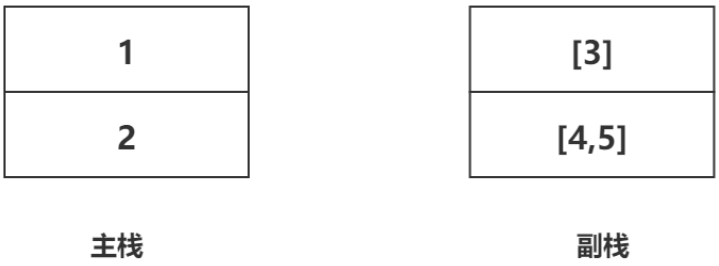

图 2-6 第二步入栈结果 2

（4）第三步，情况 1：主栈找到目标节点，获取路径。

继续进行第二步的入栈过程，直至主栈栈顶到节点 7，为目标节点，我们可以发现，输出当前栈存为数组 path：['2', '1', '3', '6', '4', '7']，2-1-3-6-4-7 即为一条路径。

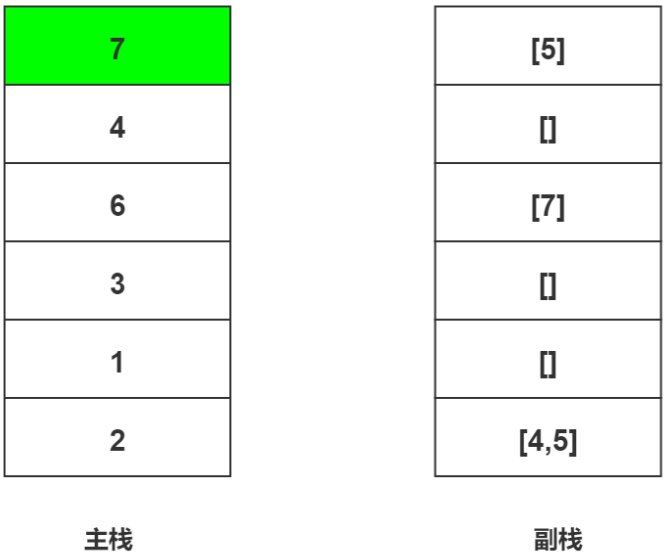

图 2-7 第一条路径的获取

同时，主栈和副栈均要做一次出栈。

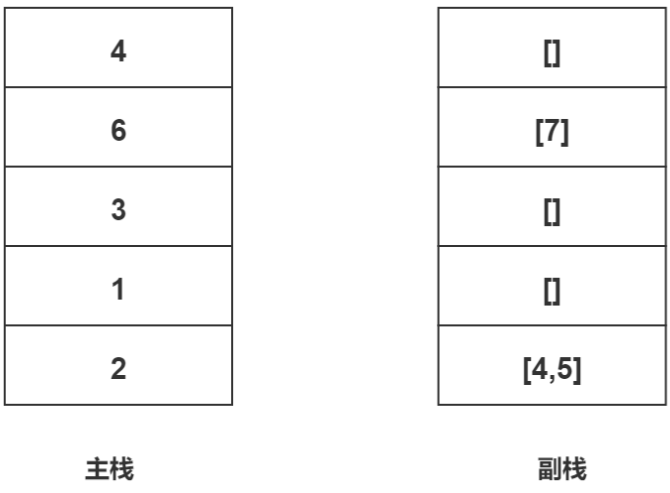

图 2-8 输出路径后双栈均出栈的结果

当然，这只是其中的一种情况，我们可以再来观察一条路径的获取过程。

（5）第三步，情况 2：副栈栈顶为空，出栈。

在前一步出栈后的结果上继续，此时，副栈的栈顶是空表[]，不能继续入栈，说明这条路径走到底仍无法找到目标节点 7。

那么，针对这一“无路可走”的情况，进行回退，查看之前的岔路。

于是，对主栈和副栈均做一次出栈。

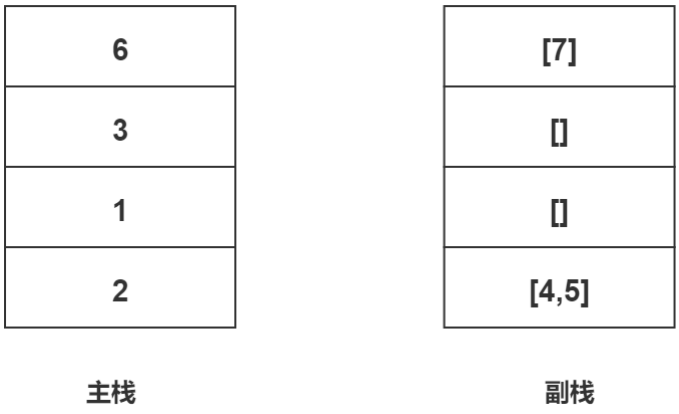

图 2-9 副栈栈顶为空，双栈均出栈的结果

接下来，查看副栈，栈顶元素非空。则继续执行第二步，取出副栈栈顶的第一个元素 7，将其压入主栈；同时将剩下的表[]压回副栈。主栈栈顶元素 7，为目标节点，我们可以发现，输出当前栈存为数组 path：['2', '1', '3', '6', '7']，2-1-3-6-7 即为另一条路径。当然，双栈也要再各做一次出栈操作。

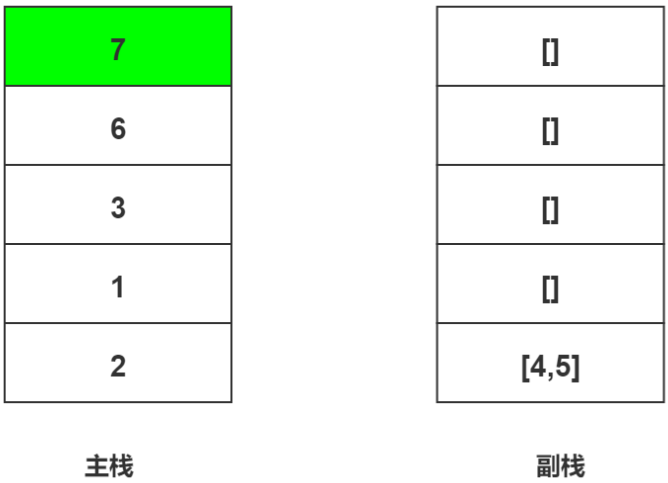

图 2-10 第二条路径的获取

（6）第四步，获取所有路径。

从上述算法的执行过程，我们可以发现第二、三步入栈和出栈的操作是重复执行的，于是添加一个循环便能获得所有路径。

第二、三步中，我们采取的手段分为三种情况：

1、副栈栈顶是非空列表，做入栈操作

2、副栈栈顶是空列表，做出栈操作

3、只要主栈栈顶是目标节点，我们输出路径，同时做出栈操作

最终，经过这一过程，便可以获得所有路径。而如果目标节点与起始节点不在同一个连通分支内，path[]数组中便不会有元素，对于这一情况特判，输出“no path”。

此函数的具体设计实现如下。

```
function findAllPaths(start1, end1){
    this.path=[]
    start = Number(start1)//此函数将对象的值转换为数字
    end = Number(end1)
    //初始化主栈
    var main_stack = new Stack()
    main_stack.push(start);
    //初始化副栈
    var side_stack = new Stack()
    side_stack.push(this.showLjBiao(start));
    var side_top//副栈栈顶元素
    var turn = 0//用于记录路径的条数（path 二维数组的维数）
    while (main_stack.length() > 0){
        //主栈不为空
        side_top = side_stack.pop()
        if (side_top.length > 0){
            //栈顶的邻接表不为空
            var main_top = side_top[0]//获取邻接表首个元素，作主栈栈顶
            main_stack.push(main_top)//将该元素压入主栈
            side_top.shift()//  移除数组首个元素
            // console.log("此时副栈栈顶为非空列表")
            side_stack.push(side_top)//剩下列表压入副栈
            if(this.showLjBiao(main_top).length > 0){
                //  如果存在下一节点
                side_top = []
                for (var i = 0; i < this.showLjBiao(main_top).length; i++){
                    var flag = 0//利用 flag 标记，用于记录邻接表中是否有元素在主栈中
                    for (var j = 0; j < main_stack.length(); j++){
                        if (this.showLjBiao(main_top)[i] === main_stack.dataStore[j]){
                            //判断是否已经有元素在主栈中
                            flag = 1
                        }
                    }
                    if (flag === 0){
                        //邻接表中没有元素在主栈中
```

```
                        side_top.push(this.showLjBiao(main_top)[i])
                    }
                }
                //剩下列表压入副栈
                side_stack.push(side_top)
            }
        }
        else{
            //栈顶的邻接表为空
            main_stack.pop()//主栈做一次出栈
        }
        var peekNode = main_stack.peek()//主栈栈顶元素
        if (main_stack.length() > 0 && peekNode === end){
            //主栈栈顶元素===目标节点
            this.path[turn] = new Array()//主栈从下至上的元素即路径
            this.sidepath[turn] = new Array()//副栈从下至上的元素
            //查看主栈
            for (var i = 0; i < main_stack.length(); i++){
                this.path[turn][i] = main_stack.dataStore[i];
            }
            //  查看副栈
            for (var i = 0; i < side_stack.length(); i++){
                this.sidepath[turn][i] = side_stack.dataStore[i];
            }
            //主栈做一次出栈
            main_stack.pop()
            if (side_stack.length() > 0){
                //副栈做一次出栈
                side_stack.pop()
            }
            turn++//路径条数加一
        }
    }
    if (turn === 0){
        console.log("no path")
```

```
            return null
        }
        else{
            return this.path
        }
}
```

至此，我们已经找到了两点之间的所有路径。距离最终的需求仅剩一步之遥，即寻找这两个点之间所有路径中最小安全值最大的路径。为此我定义了 findMinPath 函数，在 MyGraphMat.js 文件中，作为 GraphMat 类的成员函数。首先是定义了一个路径权值矩阵，将此前得到的路径对相邻两点赋予权值，使用到的数据是最之前获得的邻接矩阵。寻找安全道路遵循以下规则：

1、是否有路径

2、有路径，则按格式输出；无路径，则输出“no path”

3、minweight 的最大值不止一个时，输出其中路径上道路条数最少的一条

4、道路条数也一样，则输出终点的前驱结点序号相对小的路径

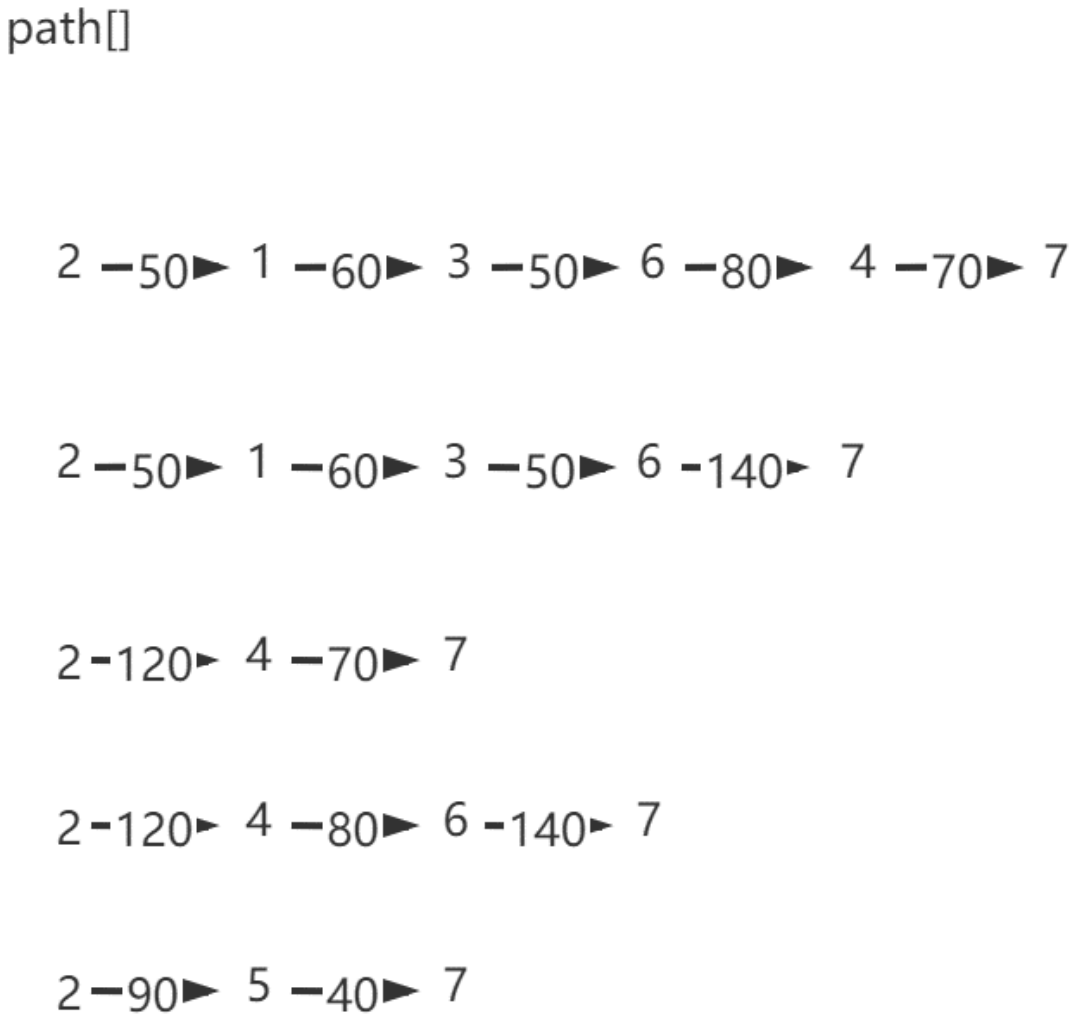

图 2-11 上例中节点 2 到节点 7 所有带权值的路径

说明：根据我们的规则，这里将会输出 80 2-4-6-7。

其函数设计如下。

```
function findMinPath(p,q){
    // 根据 p——路径，q——权值，建立新的数组存放着一条路径上的各条边
    // 最后结果得到每条路径上的所有边的权值
    for (var i = 0; i < p.length; i++){
        this.pathweight[i] = new Array()
        for (var j = 0; j < p[i].length - 1; j++){
            // -1 的原因：例如 5 个点之间有 4 条路
            var pfrom = p[i][j]
            // console.log(pfrom)
            var pto = p[i][j+1]
            // console.log(pto)
            this.pathweight[i][j] = q[pfrom][pto]
        }
    }
    // console.log(this.pathweight)
    // 对刚才得到的路径权值矩阵进行找最小值操作
    var minweight = []
    for (var i = 0; i < this.pathweight.length; i++){
        minweight[i] = MAX
        // 使用 Math 中的 max 方法
        // 使用 apply 来实现，apply 传入的是一个数组
        minweight[i] = Math.min.apply(null, this.pathweight[i]);
    }
    // 接下来通过 minweight 数组的值来确定最终选取的路径
    // 规则：
    // 1、是否有路径-----通过 path 数组是否为空来判断
    // 2、有路径，则 80：2 4 6 7；无路径，则 no path
    // 3、minweight 的最大值不止一个时，输出其中路径上道路条数最少的一条。
    // 4、道路条数也一样，则输出终点的前驱结点序号相对小的路径
    maxMW = Math.max.apply(null, minweight);
    var index = [] //记录最大值元素的下标
    for (i = 0; i < minweight.length; i++){
        if (minweight[i] === maxMW){
            // console.log(i)
            index.push(i)
```

```
        }
    }
    if (index.length === 1){
        //利用此方法将输出规整化
        let result = `${minweight[index]}: `
        result +=`${p[index].join(' ')} `;
        console.log("情况 1  最小安全值的最大值只有一个")
        console.log(result)
    }
    else if(index.length > 1){
        // minweight 不只一个时
        var pathlength = []
        for (var i = 0; i < index.length; i++){
            pathlength[i] = p[index[i]].length
        }
        minPL = Math.min.apply(null, pathlength);
        var pathindex = [] //记录最大值元素的下标
        for (i = 0; i < pathlength.length; i++){
            if (pathlength[i] === minPL){
                // console.log(i)
                pathindex.push(i)
            }
        }
        if (pathindex.length === 1){
            // minweight 不只一个，按长度再分只有一条路径
            //  先找到 pathindex 对应的 pathlength
            // pathindex 对应 index[pathindex]
            //利用此方法将输出规整化
            let result1 = `${minweight[index[pathindex]]}: `
            result1 +=`${p[index[pathindex]].join(' ')} `;
            console.log("情况 2  最小安全值的最大值不只一个，按长度再分只有一条")
            console.log(result1)
        }
        else if (pathindex.length > 1){
            var lastprenode =[]
```

```
                // 记录终点的前驱结点
                // pathindex 对应 index[pathindex]
                for (var i = 0; i < pathindex.length; i++){
                    // lastprenode[i] = new Array()
                    var len1 = p[index[i]].length
                    // 倒数第二个点需-2
                    console.log("len1"+":"+i)
                    console.log(len1)
                    lastprenode[i] = p[index[i]][len1-2]
                }
                // console.log("lastprenode")
                // console.log(lastprenode)
                minLPN = Math.min.apply(null, lastprenode);// 输出相对小的前驱节点，必定唯一
                var minLPN_node
                for (i = 0; i < lastprenode.length; i++){
                    if (lastprenode[i] === minLPN){
                        // console.log(i)
                        minLPN_node = i
                    }
                }
                // minLPN_node 即对应 index[minLPN_node]
                //利用此方法将输出规整化
                let result2 = `${minweight[index[minLPN_node]]}: `
                result2 +=`${p[index[minLPN_node]].join(' ')} `;
                console.log("情况 3 最小安全值的最大值不只一个，按长度再分不只一条，输出前驱结点序号较小的")
                console.log(result2)
            }
        }
        return minweight
}
```

2.2.4 绘制节点图例

这一功能实现是基于 d3.js 库的力导向图，利用 drawforce 函数，定义在 index 2.0.html 文件中，具体实现不在此赘述。

3．测试过程

3.1 测试题目给定的第一组样例，数据为 sample1.csv。

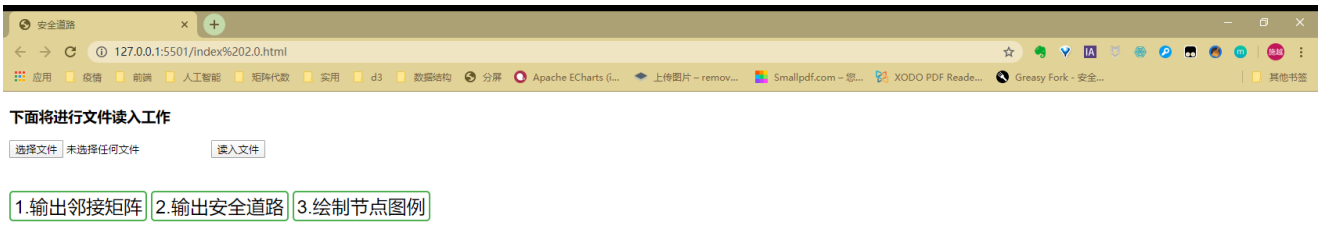

图 3-1 程序运行网页界面

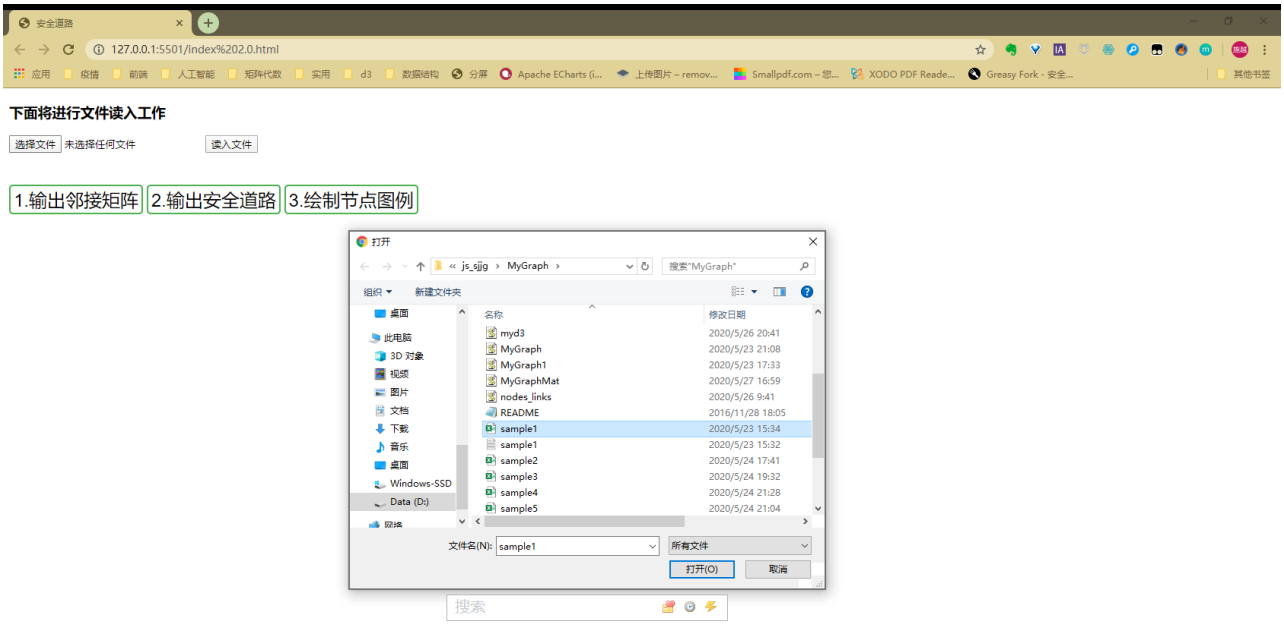

图 3-2 在左上角点击选择文件后，进行文件读入工作

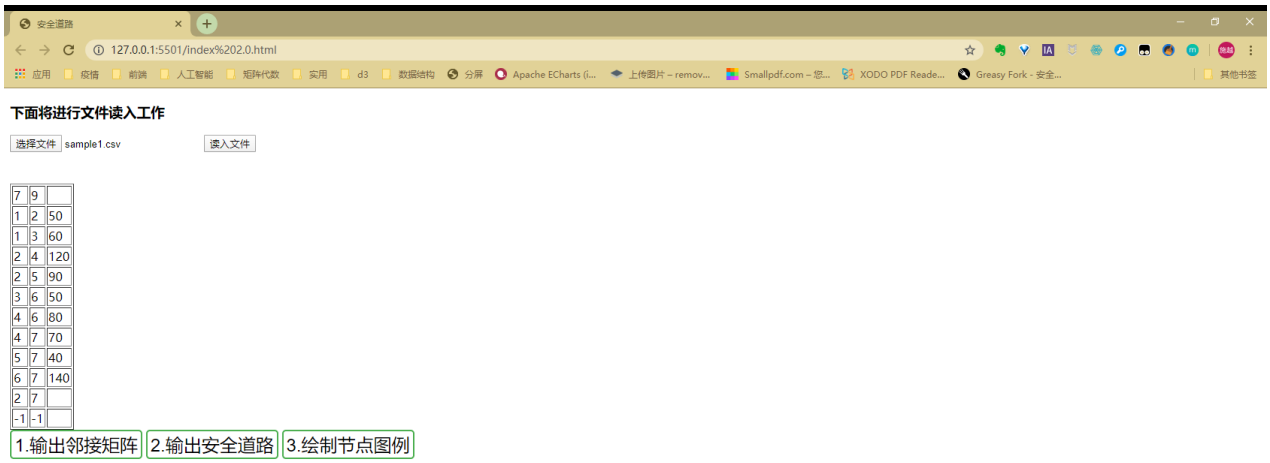

图 3-3 文件读入之后，页面左侧显示此二维数组

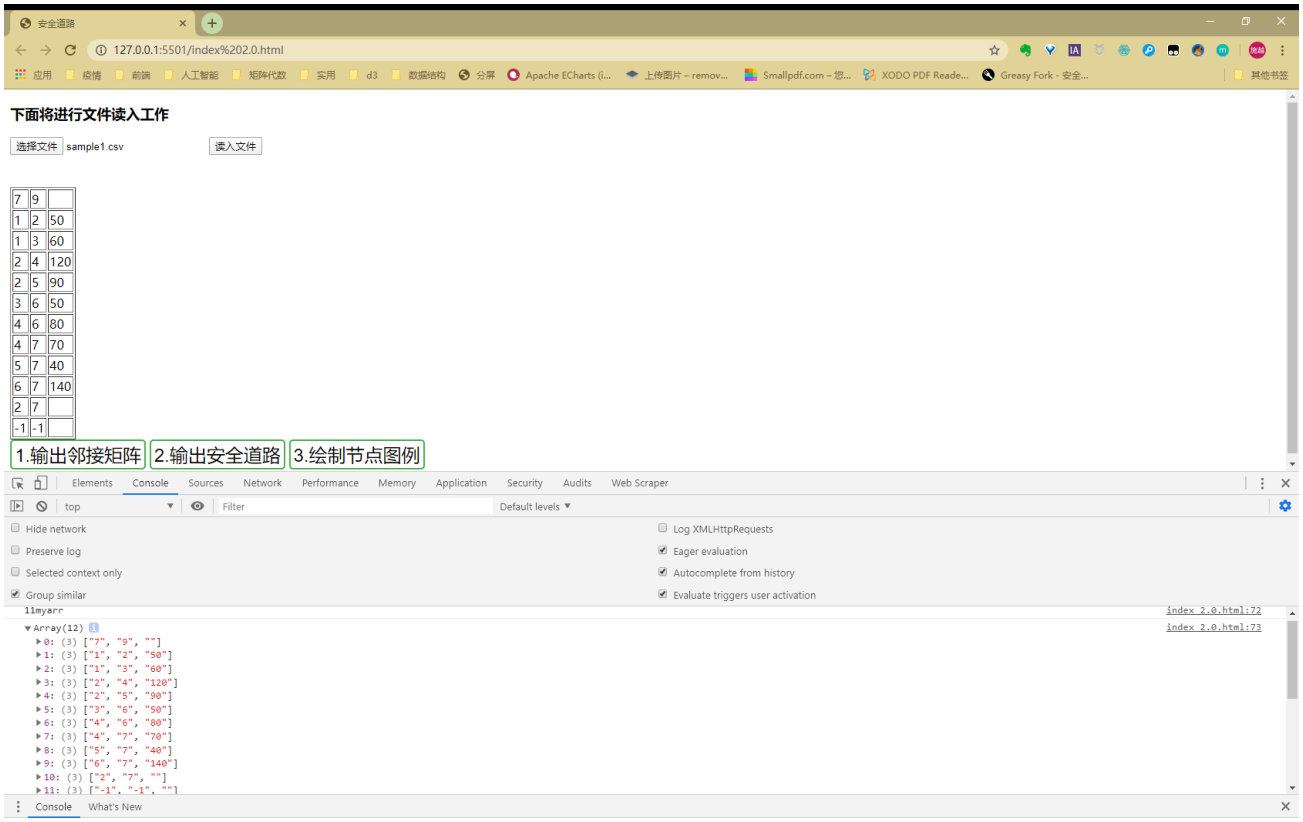

图 3-4 控制台中输出这一数组验证是否正确

下面将进行文件读入工作

选择文件 sample1.csv　　读入文件

7	9	
1	2	50
1	3	60
2	4	120
2	5	90
3	6	50
4	6	80
4	7	70
5	7	40
6	7	140
2	7	
-1	-1	

1.输出邻接矩阵　2.输出安全道路　3.绘制节点图例

图 3-5 绘制节点图例

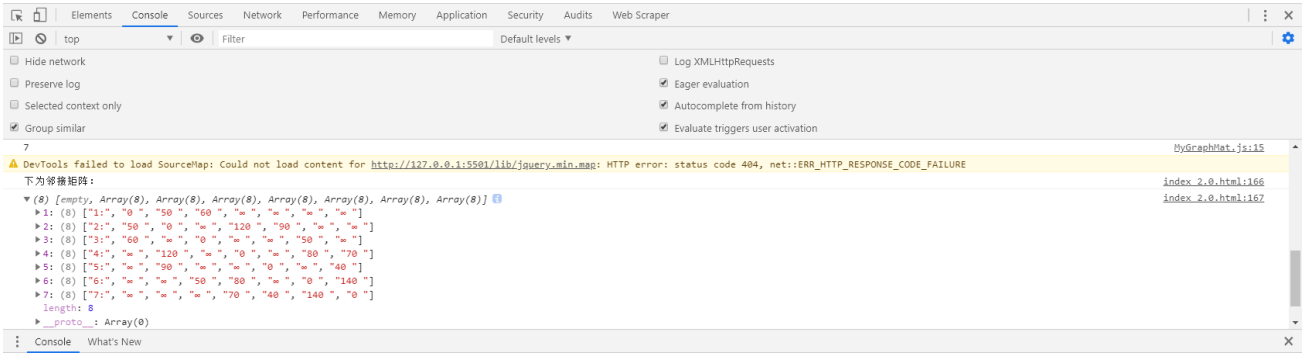

图 3-6 输出邻接矩阵

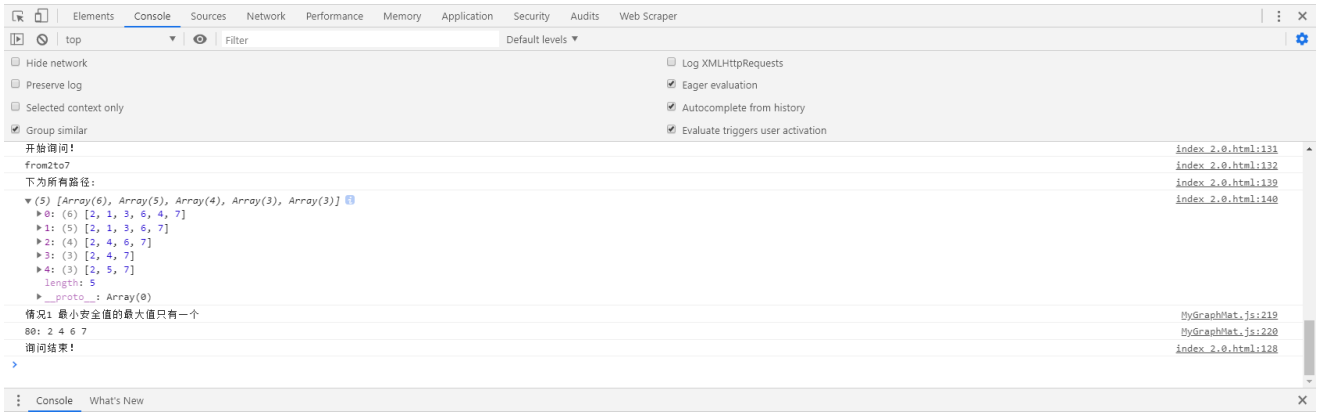

图 3-7 输出所有路径以及安全道路

3.2 测试题目给定的第二组样例，数据为 sample2.csv。部分步骤类似，不再进行。

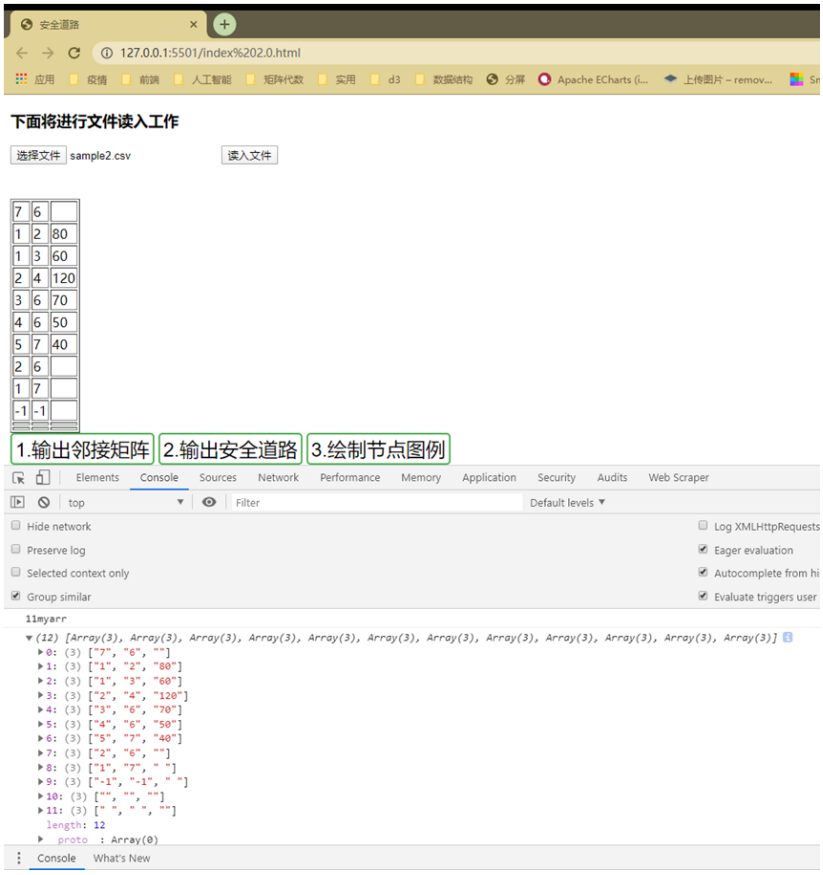

图 3-8 控制台中输出这一数组验证是否正确

图 3-9 绘制节点图例

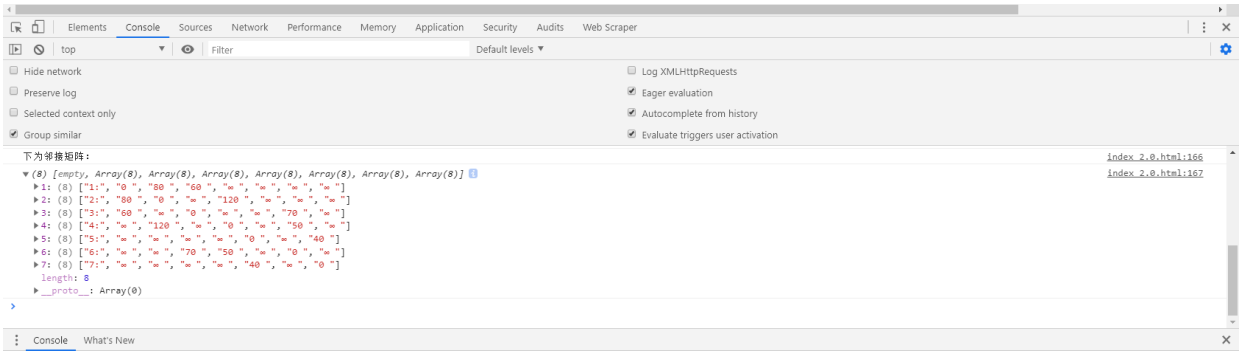

图 3-10 输出邻接矩阵

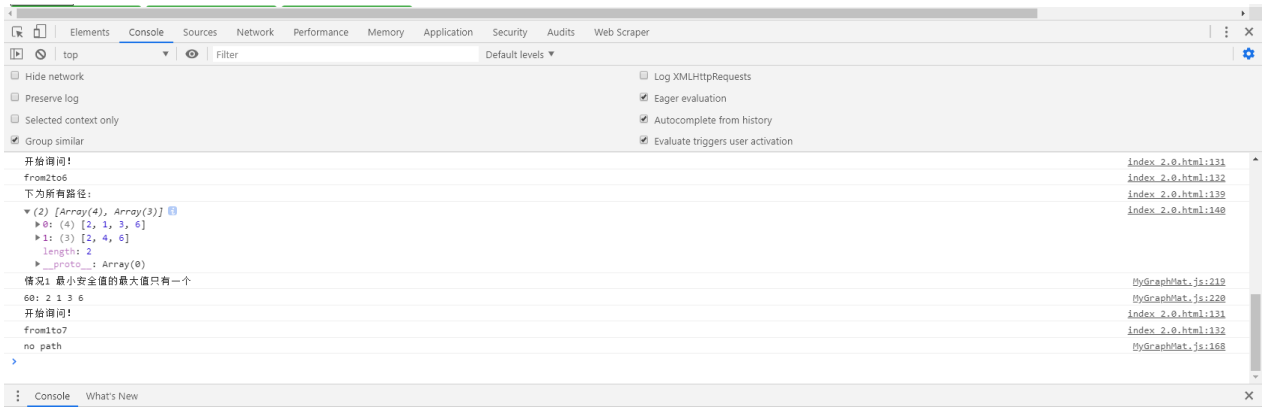

图 3-11 输出所有路径以及安全道路

3.3 测试自己给定的第三组样例，数据为 sample4.csv。

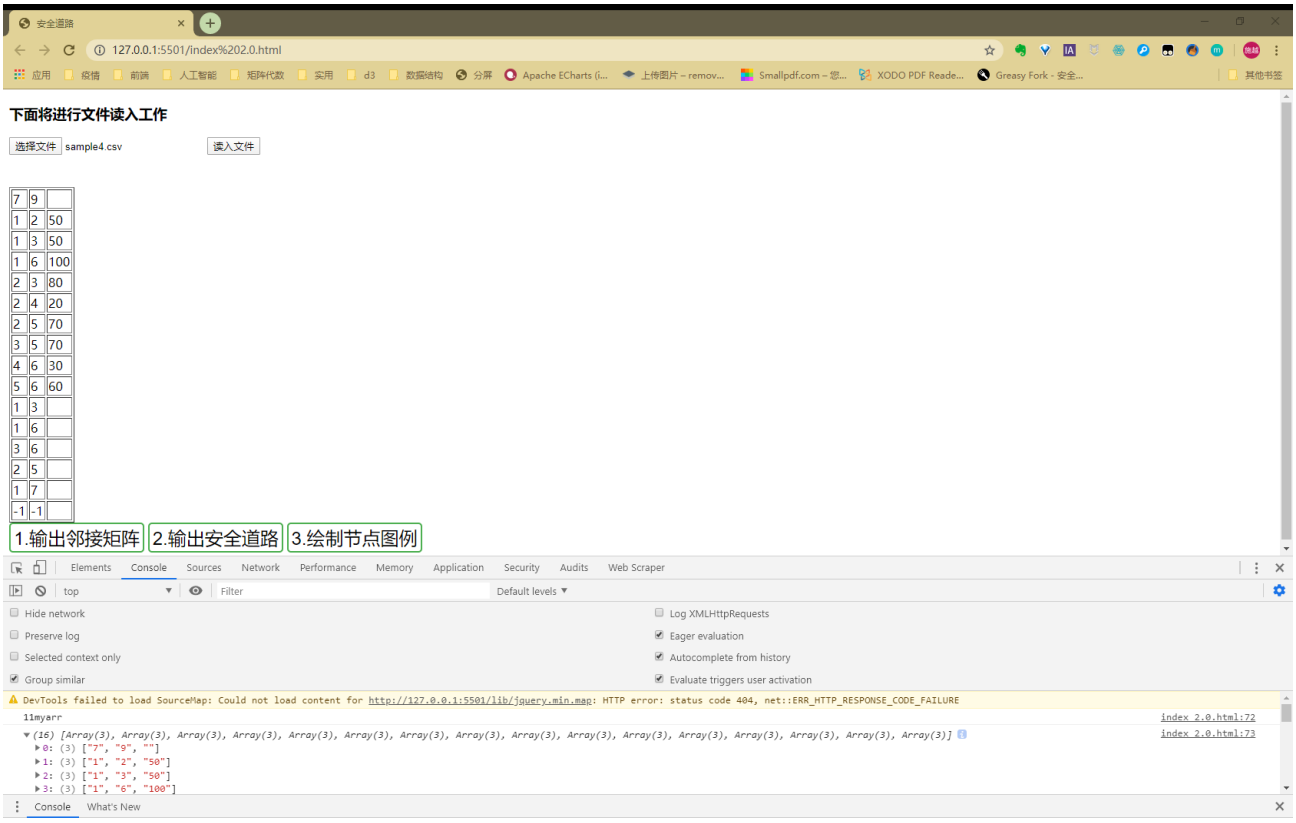

图 3-12 控制台中输出这一数组验证是否正确

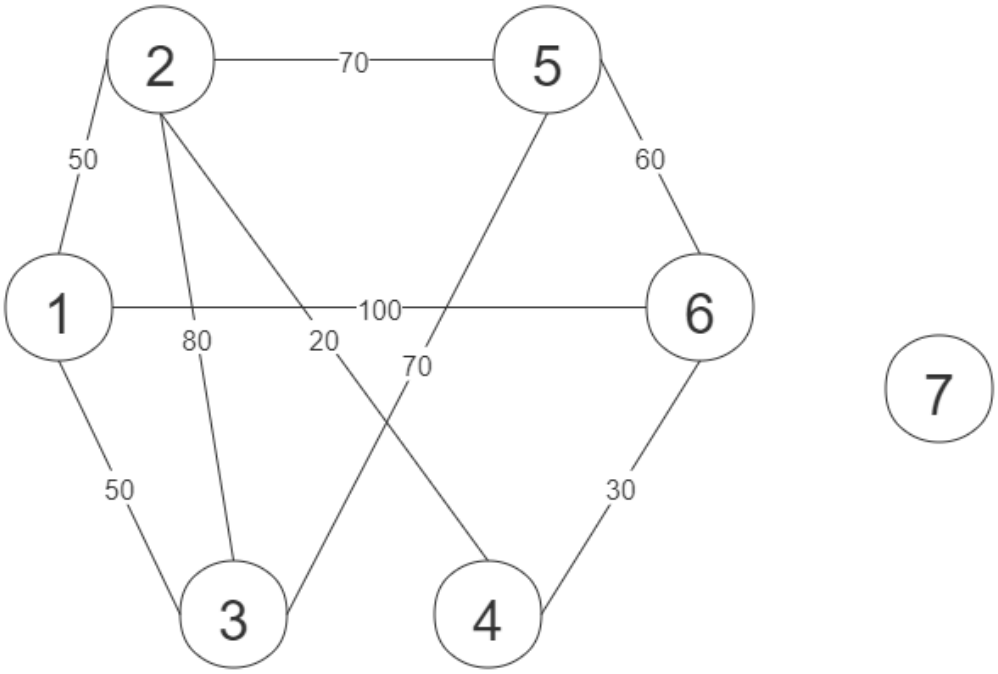

图 3-13 节点图

```
开始询问!
from1to3
下为所有路径:
▼(8) [Array(3), Array(6), Array(4), Array(2), Array(5), Array(6), Array(5), Array(4)]
  ▶0: (3) [1, 2, 3]
  ▶1: (6) [1, 2, 4, 6, 5, 3]
  ▶2: (4) [1, 2, 5, 3]
  ▶3: (2) [1, 3]
  ▶4: (5) [1, 6, 4, 2, 3]
  ▶5: (6) [1, 6, 4, 2, 5, 3]
  ▶6: (5) [1, 6, 5, 2, 3]
  ▶7: (4) [1, 6, 5, 3]
    length: 8
  ▶__proto__: Array(0)
情况2 最小安全值的最大值不只一个，按长度再分只有一条
60: 1 6 5 3
```

图 3-14 节点 1 到节点 3 的安全道路

说明：这种情况为第 2 种，最小安全值的最大值不只一个，1-6-5-3 和 1-6-5-2-3 最小安全值的最大值相同，前者路径更短，故选择 1-6-5-3。

3.4 测试自己给定的第四组样例，数据为 sample5.csv。

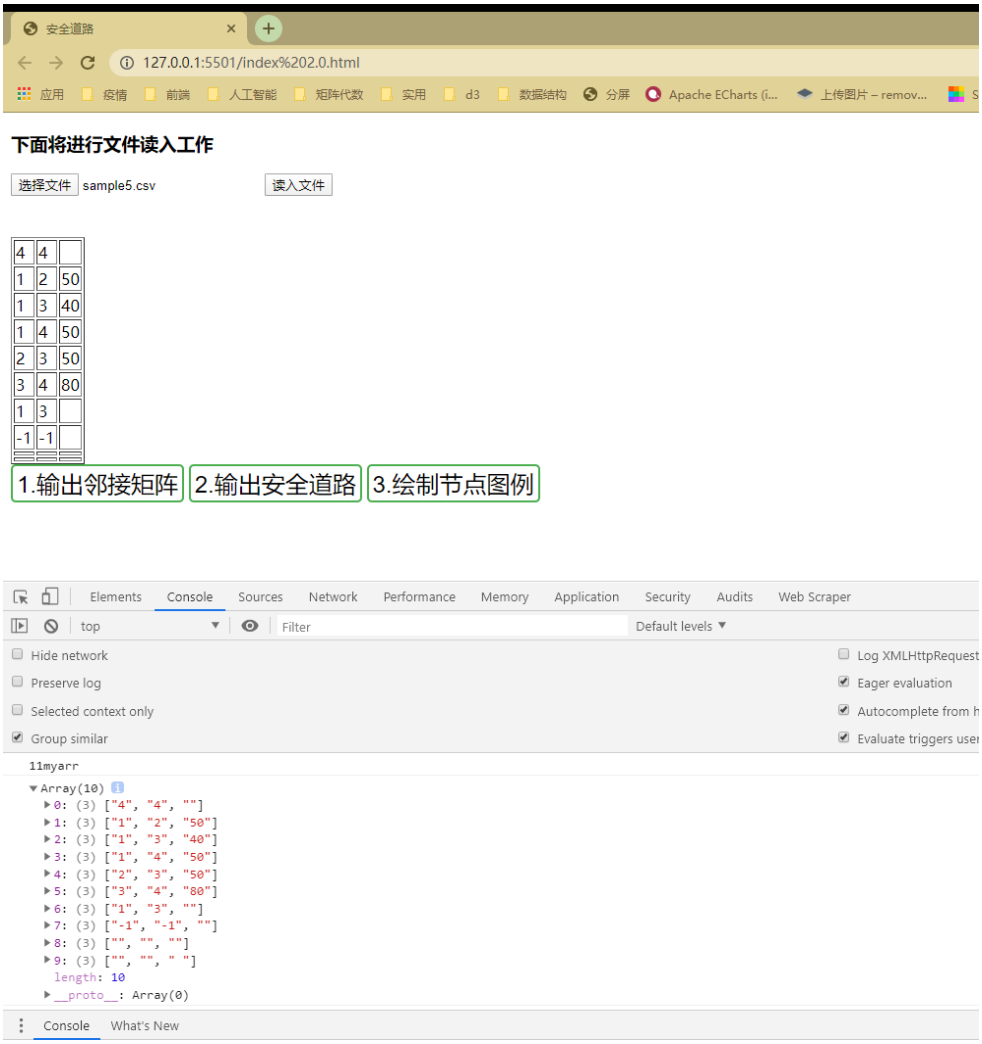

图 3-15 控制台中输出这一数组验证是否正确

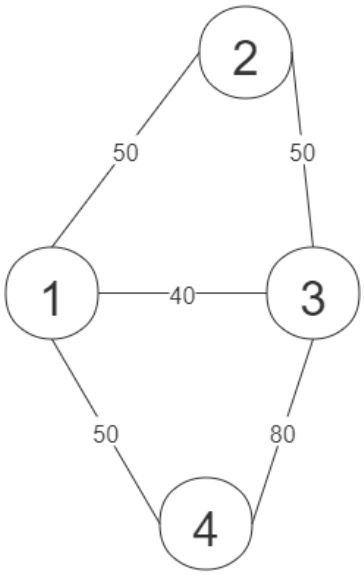

图 3-16 节点图

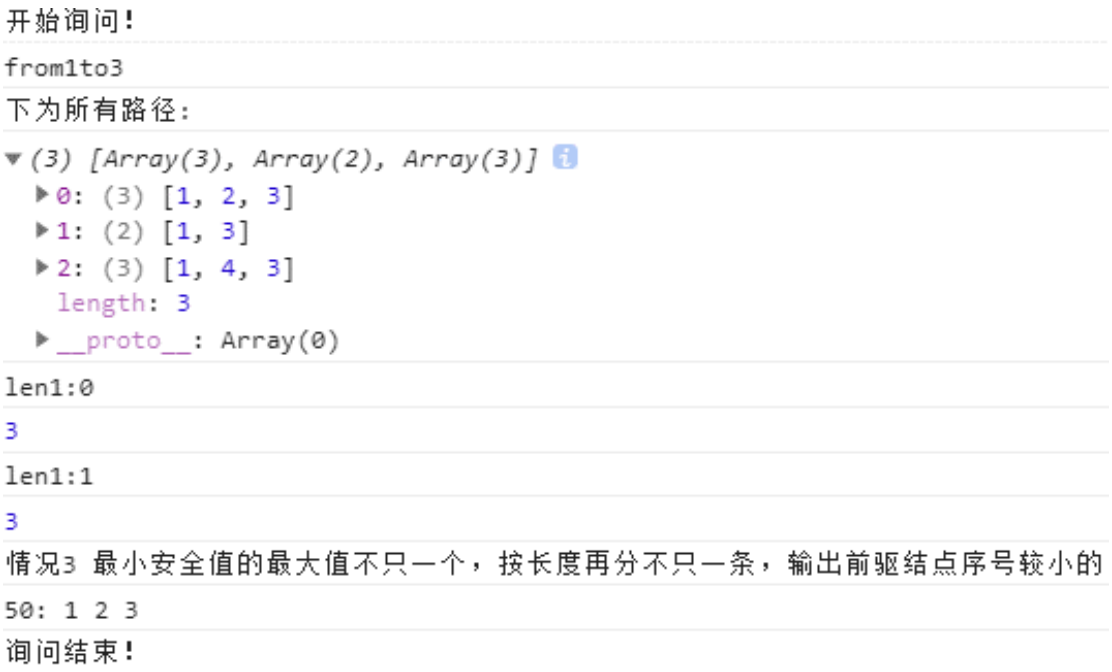

图 3-17 节点 1 到节点 3 的安全道路

说明：这种情况为第 3 种，最小安全值的最大值不只一个，1-2-3 和 1-4-3 最小安全值的最大值相同，均为 50，路径长度也相同，1-2-3 的前驱结点序号较小，故输出 1-2-3。

3.5 测试自己给定的第五组样例，数据为 sample6.csv。

图 3-18 控制台中输出这一数组验证是否正确

```
from2to100
下为所有路径:
▼(5) [Array(6), Array(5), Array(6), Array(5), Array(2)] i
  ▶0: (6) [2, 4, 6, 8, 99, 100]
  ▶1: (5) [2, 4, 6, 8, 100]
  ▶2: (6) [2, 4, 6, 99, 8, 100]
  ▶3: (5) [2, 4, 6, 99, 100]
  ▶4: (2) [2, 100]
    length: 5
  ▶__proto__: Array(0)
len1:0
6
len1:1
5
情况3 最小安全值的最大值不只一个，按长度再分不只一条，输出前驱结点序号较小的
50: 2 4 6 8 100
询问结束!
```

图 3-19 节点 2 到节点 100 的安全道路

说明：这种情况为第 3 种最小安全值的最大值不只一个，除 2-100 外，其余路径最小安全值的最大值相同，均为 50，其中 2-4-6-8-100 和 2-4-6-99-100 路径长度也相同，2-4-6-8-100 的前驱结点序号较小，故输出 2-4-6-8-100。

4．其它说明

开发语言：Html+JavaScript
开发工具：Visual Studio Code
浏览器：Chrome
插件：Live Server
文件说明：测试使用到的 csv 数据均在 P1\sample 文件夹中
csv2arr.js 文件用于转换 csv 文件
index 2.0.html 为主函数，页面呈现及调用逻辑均在其中
MyGraphMat.js 中，定义了图类，以及图的方法，是函数功能的实现部分。

二、题目 2：例句搜索

1．主要数据结构

对于本题目，我采用了 python 语言进行编写，并利用到了自然语言处理工具包 NLTK 库。接下来，我将罗列所使用的数据结构。

1.1 字典

这一数据结构体现在本题的生成字典功能中，我利用到了 python 中的字典对象，它是另一种可变容器模型，且可存储任意类型对象，其包含键和值两个属性。本题中，我将键存放单词，值则是其出现频数。具体实现将在算法设计中 2.2.2 展开。

2．主要算法设计

2.1 系统功能模块简述

主要功能模块分为“数据预处理”、“生成词典”、“单词查询”、“返回例句”、“查看同义词和反义词”、“查看词义解释”、“查看词性变换”、“运行窗口查看输出”。其功能流程图如图 4-1 所示。

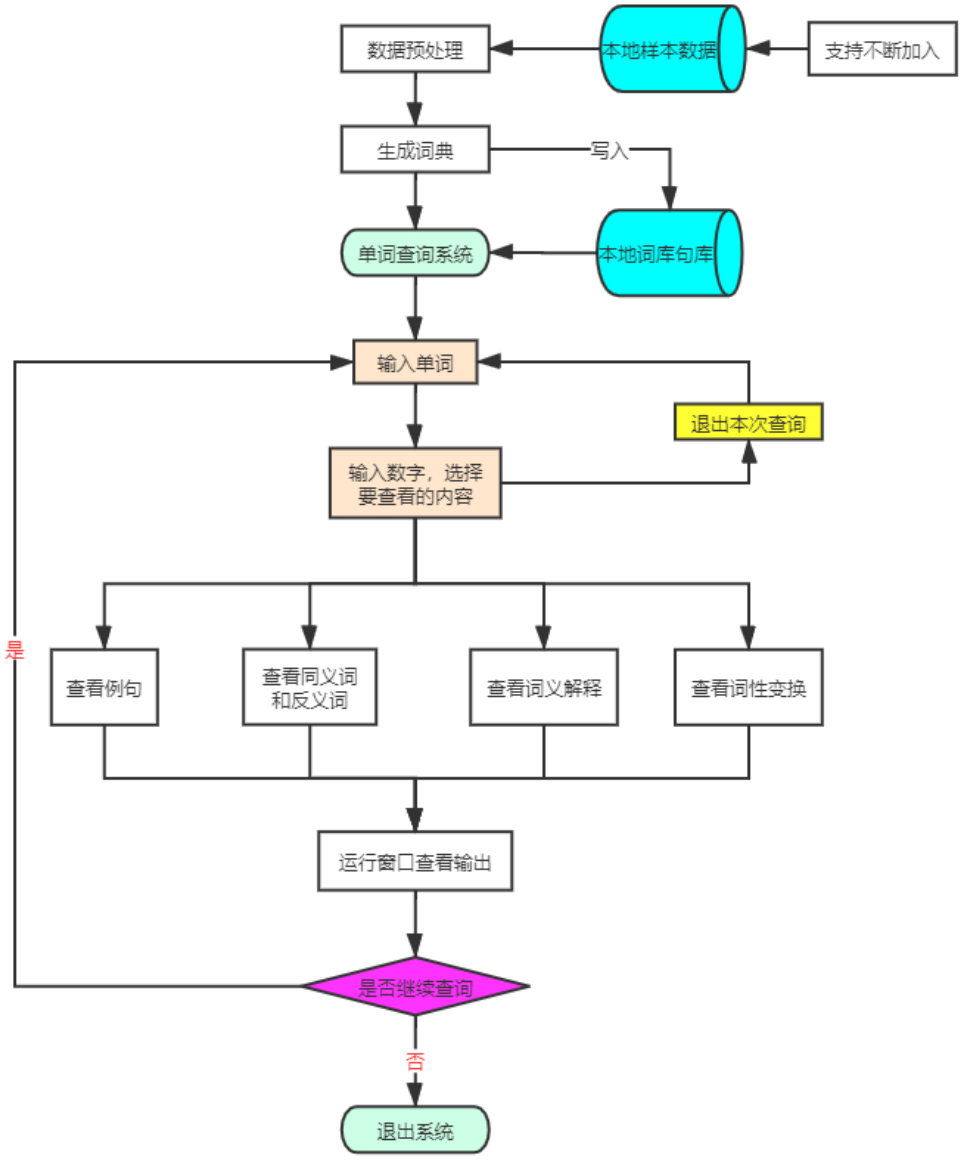

图 4-1 系统功能流程图

2.2 具体实现思路

接下来，将会对程序的各个模块的算法设计逐一进行阐述。

2.2.1 数据预处理

数据预处理的功能是将事先准备的语料进行清洗、分词操作。这里我定义了一个文件名为 Preprocess.py。在程序中，首先引入了 nltk.tokenize 模块的 sent_tokenize, word_tokenize 分别做分词和分句以及 nltk.corpus 模块的 stopwords 做清洗工作。

（1）分词

为实现分词，整体的思路大致为：首先，利用 sent_tokenize 做分句，当然这样的结果是不够的，因为其带有标点符号，而最后期望得到字典的效果。其次，便是利用正则表达式规定了模式，做适度修正：删去上一步遗留的符号。最后，利用 word_tokenize 来进行分词，英文分词相比中文分词来说更容易实现，一般只需要分隔空格即可，而 word_tokenize 方法也能够对缩写等许多特殊情况进行处理。这样一个流程下来的结果便非常理想了。

（2）清洗

清洗的形式之一是过滤掉无用的数据，在自然语言处理中，无用词（数据）被称为停止词。于是，考虑利用 nltk 语料库中的 stopwords 中的词表，选取语言为英文，使用 stop_words 集合记录这一元素集合，从文本中删除停止词的方法是做一次遍历的同时，进行判断，对于不在停止词集合中的词做一次 append 操作，添加到列表末尾。此函数的具体设计实现如下。

```
#文本预处理，做分词、清洗工作
#首先引入nltk以及re的模块，以便后续使用
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords
import re

#python正则表达式
#后续做清洗时会用到
#主要对一些标点符号进行了分割
pattern = r'(\/\w+)|(\d+\-\S+)|(\[)|(\]\S+)|(\])|(\')|(\")|(\.)|(\,)|(\;)|(\!)|(\()|(\))|\$?\d+(\.\d+)?%?| \.\.\.| ([^A-Za-z0-9]\.)+ '

#在这里增加新的语料
mytxt1=open("D:\\Y\\python\\Data_Structure\\dict\\Lincoln1.txt")#打开
sentence_list = mytxt1.readlines(100000)
open("D:\\Y\\python\\Data_Structure\\dict\\Lincoln.txt",'a+').writelines(sentence_list)

#读入准备好的文章（txt文件）
mytxt=open("D:\\Y\\python\\Data_Structure\\dict\\Lincoln.txt")#打开
s=mytxt.read()#读取
mytxt_senttok = sent_tokenize(s)#做分局
#打印利用sent_tokenize做分句后的效果
print("分句：")
```

```
print(mytxt_senttok)
#写入新的txt中
open("D:\\Y\\python\\Data_Structure\\dict\\Lincoln(after process).txt",'a+').writelines(mytxt_senttok)
#这里的属性为a+打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。也就是说，新的内容将会被写入到已有内容之后。如果该文件不存在，创建新文件进行写入。

#利用正则表达式删去标点符号
senttok_regexp = re.compile(pattern).sub('',str(mytxt_senttok))#需要转换为字符串
print("分句（正则表达式）：")
print(senttok_regexp)
open("D:\\Y\\python\\Data_Structure\\dict\\Lincoln(after regexp).txt",'a+').writelines(senttok_regexp)
#这里使用a+同理

#对此前正则化后的文本做分词
senttok_regexp_wordtok = word_tokenize(senttok_regexp)
print("分词：")
print(senttok_regexp_wordtok)

#设置停止词，做适度清洗
stop_words = set(stopwords.words('english'))
senttok_regexp_wordtok_afterstop = []
for w in senttok_regexp_wordtok:
    if w not in stop_words:
        #对不在停止词内的词做append操作，以空格分隔
        senttok_regexp_wordtok_afterstop.append(w+' ')
print("分词（去除停止词）：")
print(senttok_regexp_wordtok_afterstop)

open("D:\\Y\\python\\Data_Structure\\dict\\Lincoln_new.txt",'a+').writelines(senttok_regexp_wordtok_afterstop)
print("文本已被写入 D:\\Y\\python\\Data_Structure\\dict\\Lincoln_new.txt")
```

2.2.2 生成词典

这部分代码在 MyDictionary.py 之中，生成词典这一功能是利用此前已经过数据预处理的文本，建立索引，生成词典。我利用到了 python 中的字典对象，它是另一种可变容器模型，且可存储任意类型对象，其包含键和值两个属性。本题中，我将键存放单词，值则是其出现频数。实现思路：首先初始化字典对象，并打开一个新的 txt 文件来存储词典数据；其次读入预处理后的文本，因为此前的文本已经处理成单个词并以空格分隔的形式，这里只需要使用 split 函数进行分隔成一个列表即可，首次出现的词纳入词典中，重复的情况则无需做纳入，并将出现的频数加 1。最后只要将这一词典按照键值顺序写入 txt 文件中即可。此函数的具体设计实现如下。

```
#建立索引，生成词典
#打开之前处理完成的txt文件
f = open('D:\\Y\\python\\Data_Structure\\dict\\Lincoln_new.txt','r')
#D:\\Y\\python\\Data_Structure\\dict\\BorisJohnson.txt
o = open('D:\\Y\\python\\Data_Structure\\dict\\testDic.txt','w')  # D:\\Y\\python\\Data_Structure\\dict\\testDic.txt
dict = {}  # 初始化字典
cnt = 0
while True:
    #利用readlines函数读入文本
    sentence_list = f.readlines(100000)
    if not sentence_list:
        break
    else:
        for sent in sentence_list:
            # sent = sent.decode('utf8')
            sent = sent.split(u' ')
            for word in sent:
                if word == u'':
                    continue
                if not word in dict:
                    dict[word] = 1
                else:
                    dict[word] += 1
        for key, value in dict.items():
            # 以列表返回可遍历的(键，值) 元组数组
            # cnt += value
            # o.write(key+' %d'%value+'\n')
            # 写入文本
            o.write(key + '\n')
            # o.write('sum = %d'%cnt)
f.close()
o.close()
```

2.2.3 单词查询

在主函数中，我使用到了 input() 函数进行交互，其接受一个标准输入数据，通过这一数据与词典中的键比较，返回查询信息，单词是否在词典库中。之后根据用户输入的数字进行下一步的函数调用。此部分主函数的具体设计实现如下。

```
if __name__ == '__main__':
    print("***欢迎来到单词查询系统***")
    while True:
```

```
        k = input("请输入要查询的单词：")
        if k in dict.keys():
            # print(dict[k])
            print("找到单词："+str(k)+"，其出现频数："+str(dict[k]))
            while (key):
                key = input("***您想要查看什么信息***\n  1）查看例句 \n  2）查看同义词和反义词 \n"
                            "  3）查看词义解释 \n  4）查看词性变换 \n  0）退出本次查询 \n  请输入数字：")
                if key == "1":
                    print("正在寻找中，请稍候...")
                    returnInstaSen(k)
                elif key == "2":
                    print("正在寻找中，请稍候...")
                    returnSynsandAnts(k)
                elif key == "3":
                    print("正在寻找中，请稍候...")
                    returnDefinition(k)
                elif key == "4":
                    print("正在寻找中，请稍候...")
                    returnCixin(k)
                elif key == "0":
                    break
                else:
                    print("***输入格式错误***")
        else:
            print("***字典库中未找到这个单词***")
```

2.2.4 返回例句

对于返回例句，我定义了 returnInstaSen 函数，先对于我们原始的语料即句库，进行逐行遍历返回其行数，用到了__next__() 方法；接下来做一个按照传入的 word 逐行做一个匹配工作，若此行含有该 word 便输出，倘若直至文件遍历完仍没有例句，则返回“没有找到例句”。此函数的具体设计实现如下。

```
def returnInstaSen(word):
    #返回例句
    #获取文件行数
    with open("D:\\Y\\python\\Data_Structure\\dict\\Lincoln.txt", 'r') as fo:
        length = len(fo.readlines())
        # print(length)


    fo = open("D:\\Y\\python\\Data_Structure\\dict\\Lincoln.txt", "r+")
```

```
    # word = "apple"
    cnt = 0
    for index in range(length):
        line = fo.__next__()
        if word in line:
            cnt += 1
            print("在语料库中找到例句"+str(cnt)+"：  "+line)
        if (index == length - 1 and cnt == 0):
            print("没有找到例句")
        # print("第  %d  行  - %s" % (index, line))
    fo.close()
```

2.2.5 查看同义词反义词

这里用到 nltk 中的 WordNet 的词汇数据库，定义了两个空列表存放同义词及反义词，并使用 append 函数添加到列表尾，对于空列表输出没有找到的信息。其函数设计如下。

```
def returnSynsandAnts(word):
    #寻找同义词和反义词
    synonyms = []
    antonyms = []

    for syn in wordnet.synsets(word):
        for l in syn.lemmas():
            synonyms.append(l.name())
            if l.antonyms():
                antonyms.append(l.antonyms()[0].name())
    if(set(synonyms).__len__() != 0):
        print("***下为"+str(word)+"的同义词***")
        print(set(synonyms))
    else:
        print("***没有找到"+str(word)+"的同义词***")
    if (set(antonyms).__len__() != 0):
        print("***下为"+str(word)+"的反义词***")
        print(set(antonyms))
    else:
        print("***没有找到"+str(word)+"的反义词***")
```

2.2.6 查看词义解释

同样利用 WordNet 模块，这里返回的词义是其同义词列表第一个量的词义。其实现如下。

```
def returnDefinition(word):
    #利用wordnet模块
```

```
    #该句将返回同义词列表第一个量的词义
    print(wordnet.synset(str(word)+'.n.01').definition())
```

2.2.7 查看词性变换

这里使用到的是 WordNetLemmatizer，利用 lemmatize 找到单词的名词原形。

```
def returnCixin(word):
    #利用WordNetLemmatizer模块
    lemmatizer = WordNetLemmatizer()
    #这里做的是词形还原，将输出其原形（如输入cats，将输出cat）
    print(lemmatizer.lemmatize(word))
```

3．测试过程

3.1 数据预处理

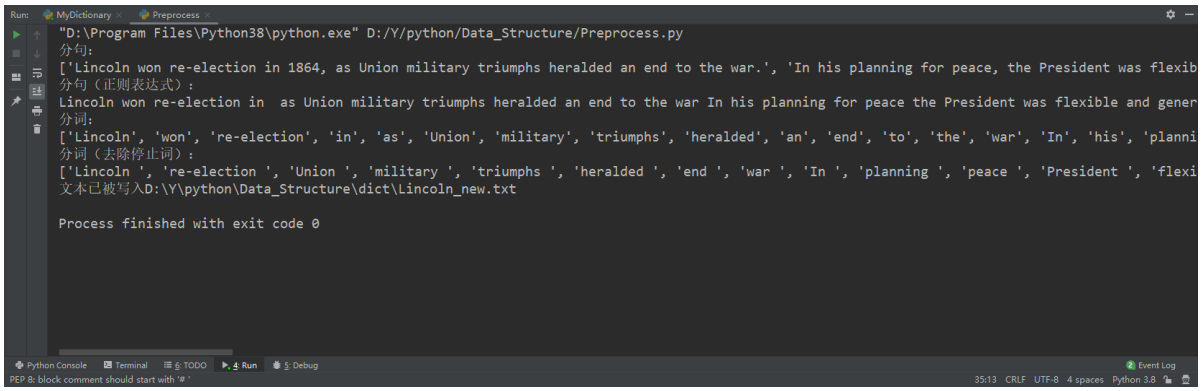

图 5-1 数据预处理效果

说明：可以看到分别利用分句模块、正则表达式、分词以及去除停止词处理的结果，最终能够得到单个的词语，用于之后生成词典。

3.2 生成词典

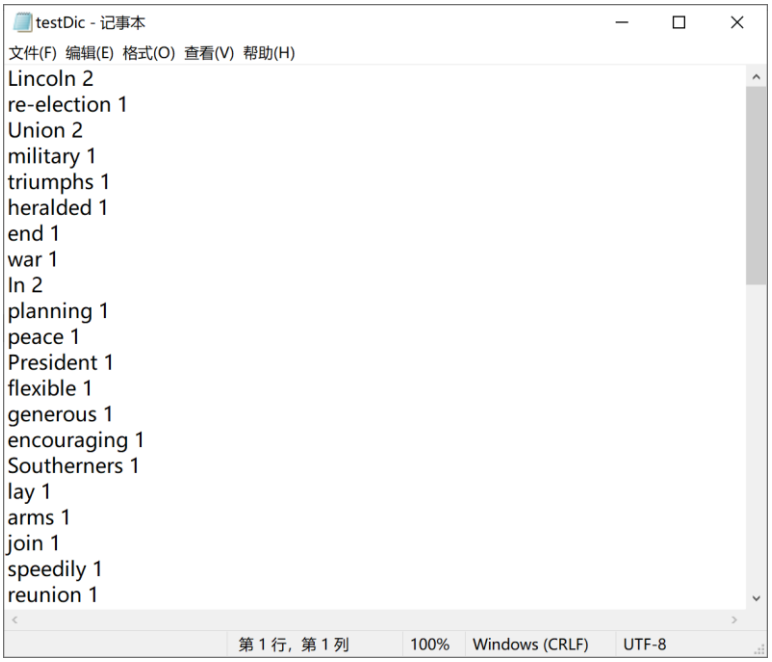

图 5-2 根据预处理后的数据生成的词典

3.3 查看词“Lincoln”信息

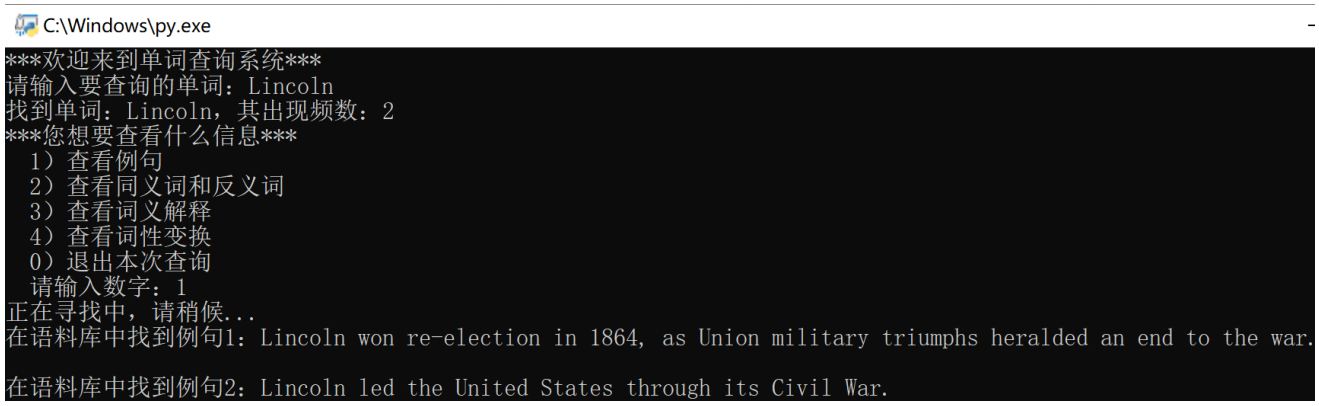

图 5-3 查看词“Lincoln”的例句

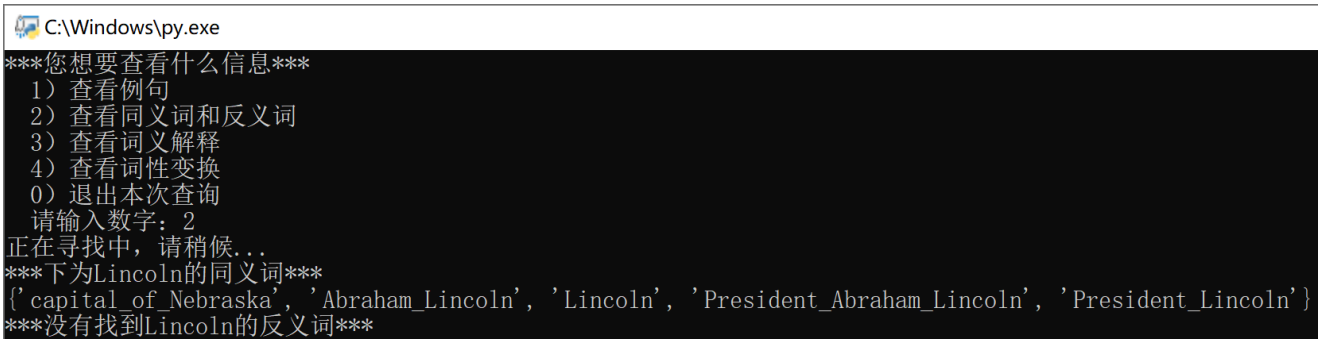

图 5-4 查看词“Lincoln”的同义词和反义词

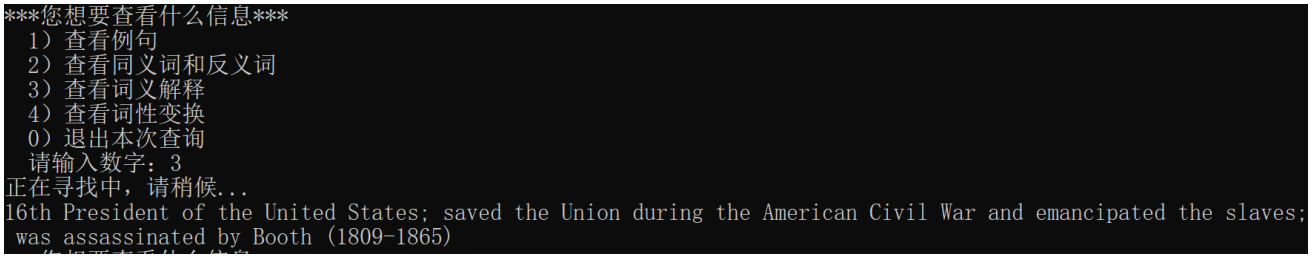

图 5-5 查看词“Lincoln”的词义解释

图 5-6 查看词“Lincoln”的词性变换

3.4 查看词“arms”的信息

```
***您想要查看什么信息***
  1）查看例句
  2）查看同义词和反义词
  3）查看词义解释
  4）查看词性变换
  0）退出本次查询
  请输入数字：0
请输入要查询的单词：arms
找到单词：arms，其出现频数：1
***您想要查看什么信息***
  1）查看例句
  2）查看同义词和反义词
  3）查看词义解释
  4）查看词性变换
  0）退出本次查询
  请输入数字：1
正在寻找中，请稍候...
在语料库中找到例句1：In his planning for peace, the President was flexible and generous, encouraging Southerne
rs to lay down their arms and join speedily in reunion.
```

图 5-7 退出本次查询，查看词“arms”的例句

```
***您想要查看什么信息***
  1）查看例句
  2）查看同义词和反义词
  3）查看词义解释
  4）查看词性变换
  0）退出本次查询
  请输入数字：2
正在寻找中，请稍候...
***下为arms的同义词***
{'coat_of_arms', 'arm', 'branch', 'weapon', 'munition', 'weapon_system', 'sleeve', 'build_up', 'gird', 'blazon
', 'blazonry', 'weapons_system', 'implements_of_war', 'limb', 'fortify', 'subdivision', 'weaponry', 'arms'}
***下为arms的反义词***
{'disarm'}
```

图 5-8 查看词“arms”的同义词和反义词

```
***您想要查看什么信息***
  1）查看例句
  2）查看同义词和反义词
  3）查看词义解释
  4）查看词性变换
  0）退出本次查询
  请输入数字：3
正在寻找中，请稍候...
weapons considered collectively
***您想要查看什么信息***
  1）查看例句
  2）查看同义词和反义词
  3）查看词义解释
  4）查看词性变换
  0）退出本次查询
  请输入数字：4
正在寻找中，请稍候...
arm
```

图 5-9 查看词“arms”的词义解释和词性变换

3.5 查看新增语料的效果

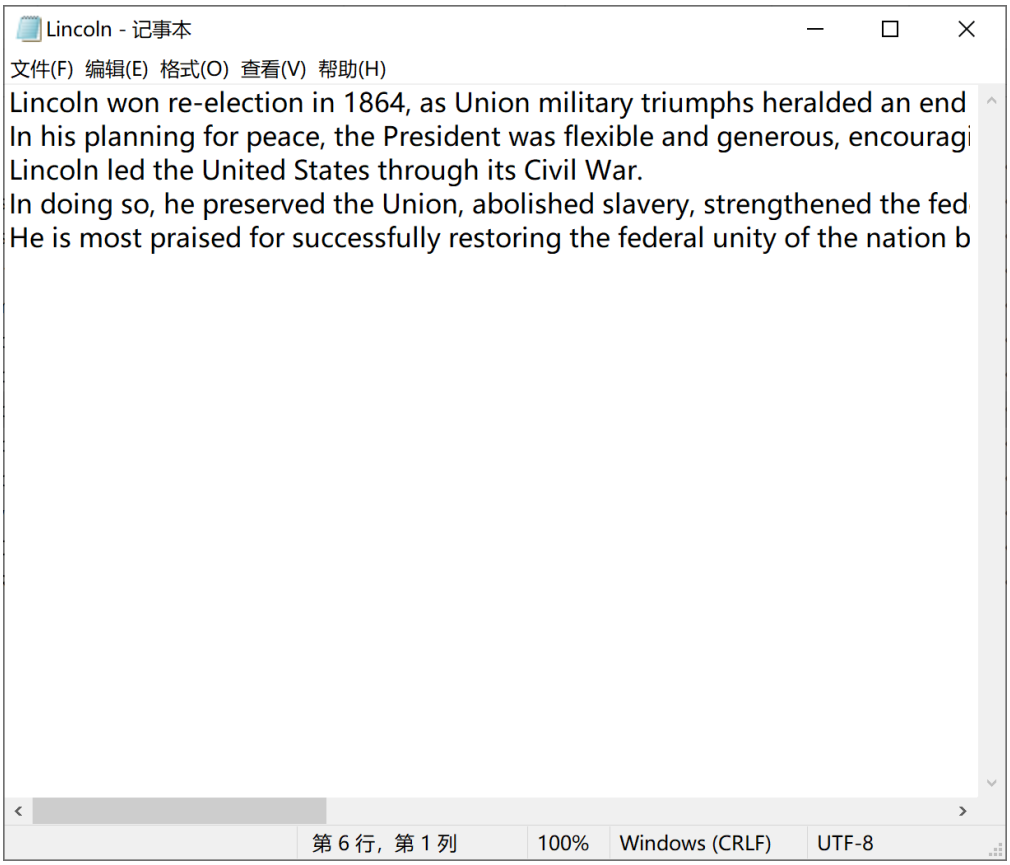

图 5-10 原语料库

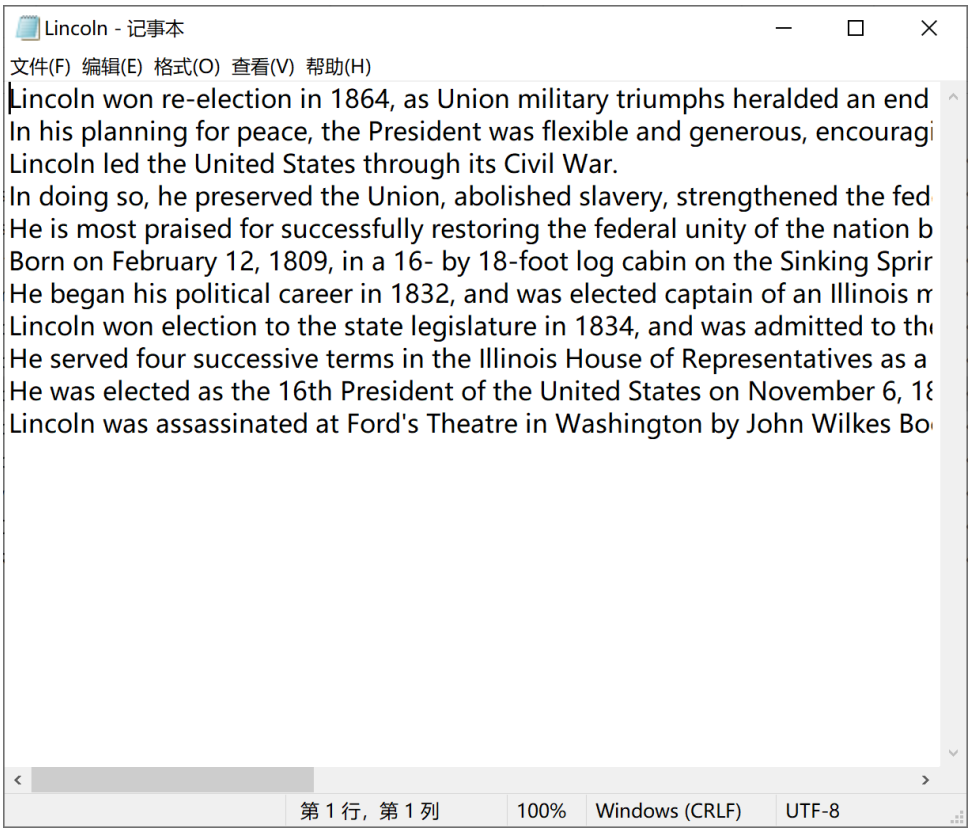

图 5-11 读入 Lincoln1.txt 后的新语料库

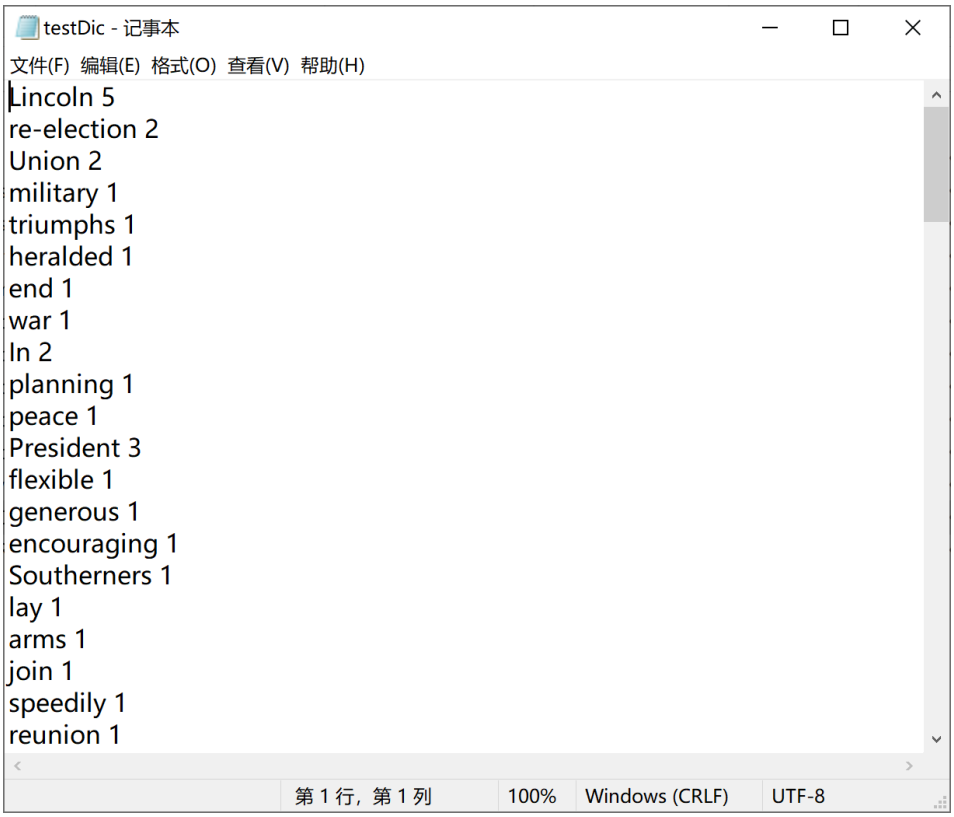

图 5-12 根据新语料库生成的词典

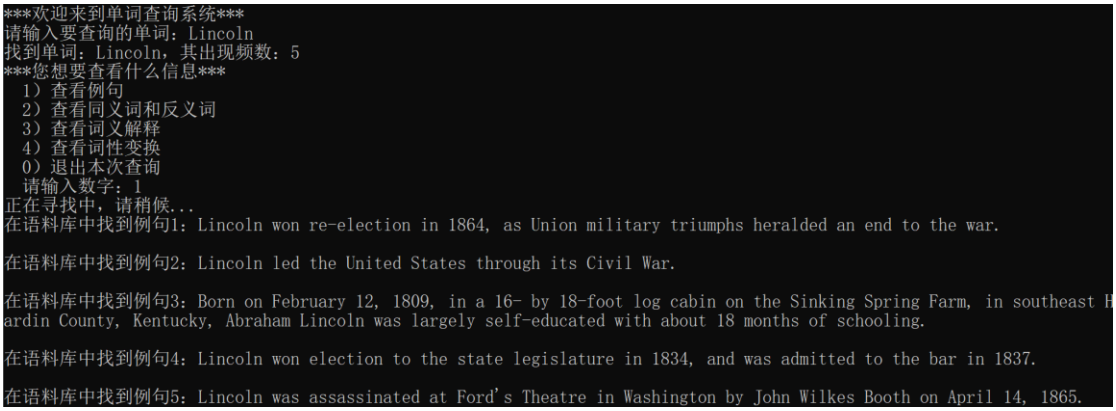

图 5-13 在新词典中查询词“Lincoln”的结果

4．其它说明

开发语言：Python
开发工具：Pycharm
工具包：NLTK
文件说明：测试使用到的 txt 数据均在 P1\dict 文件夹中
Lincoln.txt 为原始语料库
Lincoln_new.txt 为经过分词、清洗后的语料
testDic.txt 为根据新的语料生成的词典
Preprocess.py 文件用于做数据预处理工作
MyDictionary.py 文件用于生成词典以及查看信息

三、课程总结

19-20 学年冬季学期在疫情的影响之下，终于在本周结束了，《数据结构（2）》课程也迎来了最终的考核。回顾这十周包括寒假期间、在家上网课期间，对于本门课程的认识加深了不少，关于常用的数据结构也能够较为熟练的掌握。对此，我有几点感触，将在下文中展开叙述。

首先，关于如何分析问题，在这次的报告撰写过程中，一开始要做的便是选取数据结构，以及后续的算法设计，这些都是需要去进行斟酌的。就拿第一题来说，不同于平常所考虑的两点之间所有路径的问题，还要考虑权值的最小值以及后续比较的因素。但其仍然是基于基础的数据结构的，所以还是要从基本出发。在经过调研和不断尝试之后，最后选择了双栈加上循环的思路。

其次，关于如何学习数据结构。对于我们大学生来说，掌握学习方法是相当重要的。对于我所从事的项目，必定会有类似的其他项目，这时候他人的经验就显得很有参考价值了。通过论坛，书籍，博客等方式，收集到他人对于其实现方式的分享，对于自己如何考虑也能够有更好的认识，更能快速上手。对于不懂的问题不耻下问，再加以自己的思考，能够更快地获得答案。以上两点也正是我的做法，很好地帮到了我。

最后，说说自己的收获。这种考核方式对于我来说是不小的挑战，但也是通过这一周的时间，我对于如何写一个完整的项目有了一些实战经验，针对自然语言处理也有了一些浅显的认识。对于如何撰写报告（包括今后的论文），我积累了一些经验，要尽可能做到言之有物，结合一些图表，完整清晰地展示自己所做的工作，并表明自己分析问题的思路和方法。

另外，由于本人的能力有限，在这份报告中，也可能会有不少纰漏和不严密的地方，敬请郑老师批评指正。虽然这次对于某些知识的运用和衔接还不够熟练，但是我将在今后的学习中继续努力、不断完善。

最后的最后，感谢郑老师的辛勤付出，以及在我学习数据结构的过程中给予的帮助。