

---

## 《数据结构》课程设计总结

学号： 1652270

姓名： 冯舜

专业： 计算机科学与技术

2018 年 7 月

---

# 目录

第一部分	算法实现设计说明	4
1.	题目	4
2.	软件功能	4
3.	设计思想	4
3.1.	图形用户界面的设计思想	4
3.2.	数据结构和算法的设计思想	6
4.	逻辑结构与物理结构	6
4.1.	逻辑结构	6
4.2.	物理结构	7
5.	开发平台	9
6.	系统的运行结果分析说明	9
6.1.	开发、调试过程	9
6.2.	开发成果	10
6.3.	创建新树	10
6.4.	添加键值	11
6.5.	查找键值	12
6.6.	删除键值	13
7.	操作说明	14
7.1.	基本操作说明	14
7.2.	树的显示、查看	15
第二部分	综合应用设计说明	17
1.	题目	17
2.	软件功能	17
3.	设计思想	17
3.1.	数据结构设计思想	17

---

3.2.	图形界面设计思想.....	19
4.	逻辑结构与物理结构.....	20
4.1.	必要的说明.....	20
4.2.	地图数据结构.....	20
4.3.	有向图数据结构.....	23
5.	开发平台.....	24
6.	系统的运行结果分析说明.....	24
6.1.	开发及调试过程.....	24
6.2.	开发成果.....	24
6.3.	运行结果.....	25
7.	操作说明.....	28
7.1.	地图浏览.....	28
7.2.	添加站点.....	29
7.3.	添加线路.....	30
7.4.	查看线路图例、移除线路.....	31
7.5.	清除、保存、加载.....	31
7.6.	寻路.....	32
第三部分	实践总结.....	33
1.	所做的工作.....	33
2.	总结与收获.....	33
第四部分	参考文献.....	33

---

# 第一部分 算法实现设计说明

## 1. 题目

从空树出发构造一颗深度至少为 3（不包括失误结点）的 3 阶 B-树（又称 2-3 树）并可以随时进行查找、插入、删除等操作。

要求：能够把构造和删除过程中的 B-树随时显示输出来，能给出查找是否成功的相关信息。

## 2. 软件功能

本软件作为数据结构中“B-树”的实现和演示软件，做到以下功能：

- 在软件运行时内部维护一个“B-树”的数据结构，可以对其进行增加、修改、删除结点的操作，并在程序运行完成后自行释放；
- 软件应实现一个 GUI（Graphic User Interface，图形用户界面）能够适时显示 B-树经过改动后的结构。
- 软件的 GUI 应提供给用户对于 B-树进行上述操作的接口（按钮、文本框），以便用户输入。用户通过输入给出指令后，软件应执行用户的指令，并在对 B-树修改后给出显示反馈。

## 3. 设计思想

本软件的体系结构为 MVC（Model-View-Controller）式，具体分“图形用户界面”和“内部数据结构和算法”两部分。

### 3.1. 图形用户界面的设计思想

图形用户界面包括 Model 和 Controller 的部分。

由于本软件基于 Qt 平台，因此可以利用 Qt 提供的图形用户界面库中的类（QWidget 等）来实现图形界面。

本软件设计时优先考虑 Microsoft Windows 等 PC 平台，故 GUI 的设计是窗口式的。GUI 包括：

- 提供给用户的指令按钮；
- 指令配套的参数的输入框；

- 实时显示树结构的显示区域。

在 Qt Creator 中设计的 GUI 窗体如图所示。

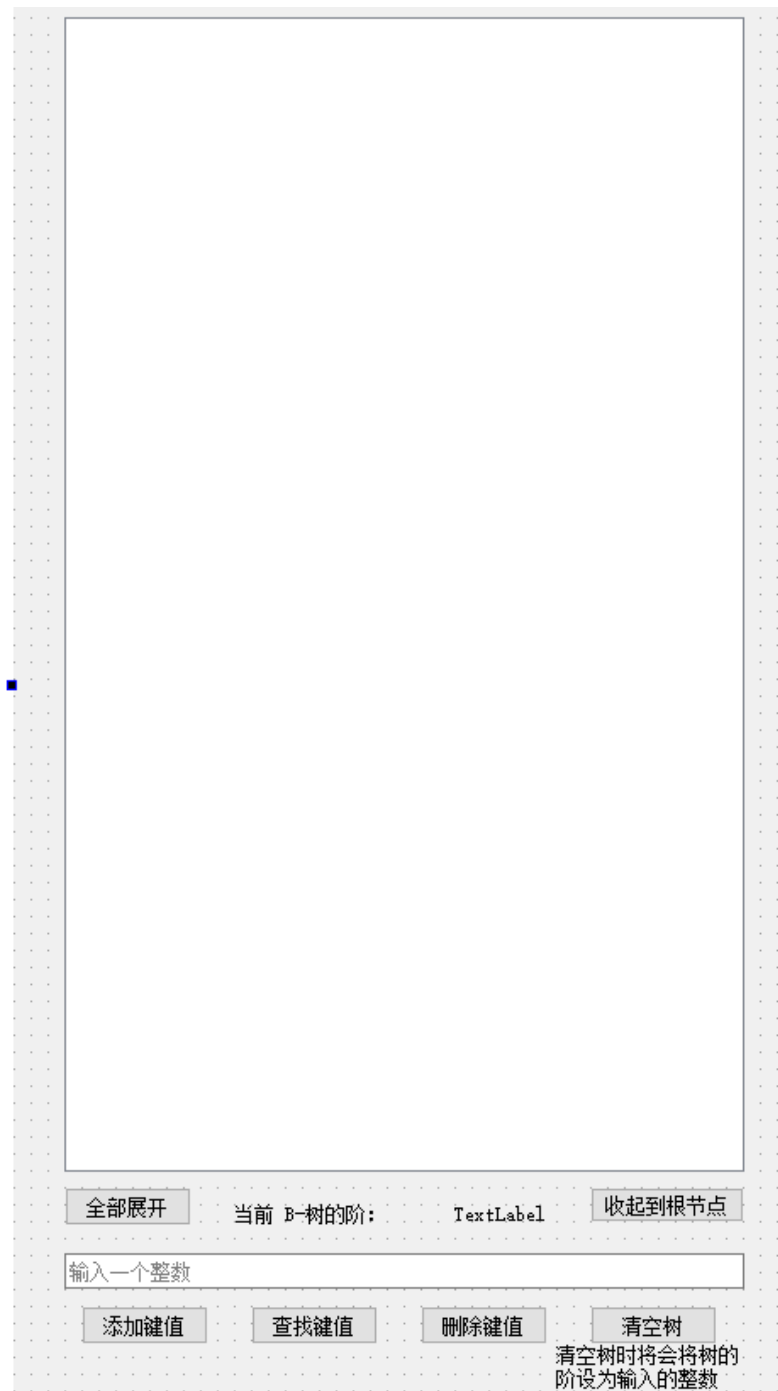


图 GUI 窗体设计

对于 B-树的打印，由于显示空间有限而树的构造可以无限，故不采用直接的图形来打印 B-树。Qt 提供了一个名叫 QTreeView 的控件，能以类 Windows 下文件目录结构的方式展示树结构的内容，具有层次性强、延展性强、易于实现的特点，故采用 QTreeView

来显示树。

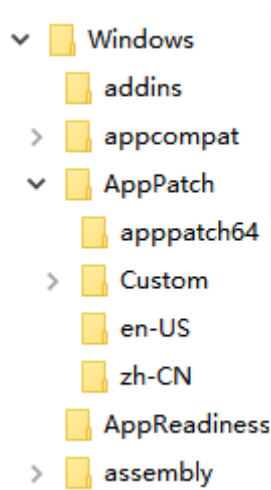


图 Windows 文件目录的显示，可展示树的内容。

## 3.2. 数据结构和算法的设计思想

本软件只涉及 B-树这一种数据结构。

在 B-树上，有“添加键值”、“删除键值”、“查找键值”三个算法。其中，添加键值算法用到了结点分裂算法、删除键值算法用到了结点合并以及关键字不足的结点的补全算法。它们均为纯粹基于 B-树的算法。

除此之外，为使 GUI 能打印该 B-树，另有一个将 B-树映射为 QTreeView 下具有层次的 QStandardItem(Model)对象的显示算法。

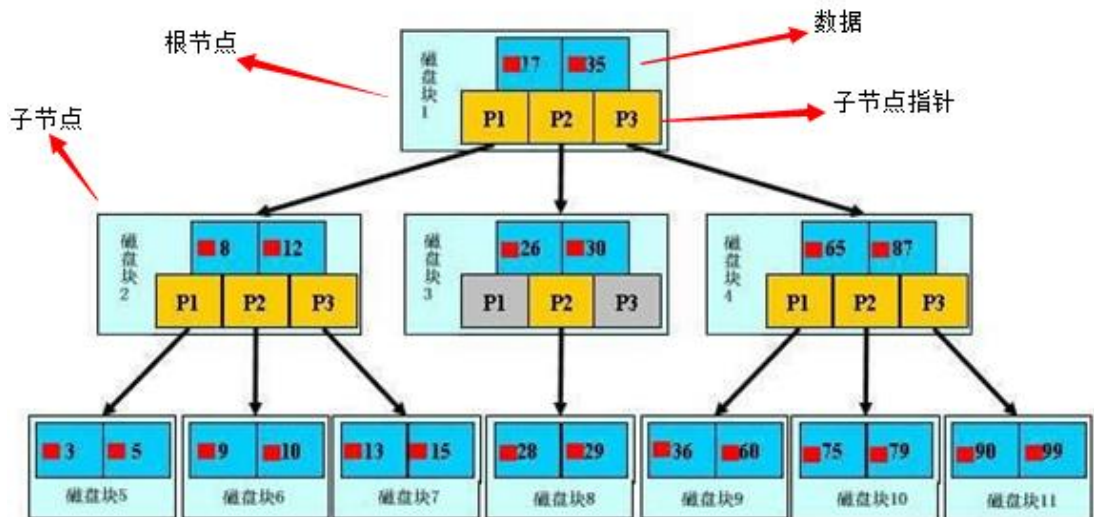
## 4. 逻辑结构与物理结构

### 4.1. 逻辑结构

B-树为多路平衡查找树，与二叉搜索树的不同在于：

- 它是多路的，这要求它是多叉树（最大分支数取决于阶）且它每两个分叉之间需要有一个搜索键值用来定位。故  $n+1$  叉结点存  $n$  个键值。
- 它的空间利用率相比较低，因为每个结点的分支数不一定达到阶（最大分支数） $m$ ，其可能值在  $m/2 + 1$  和  $m$  之间（根节点不受最小值限制）。
- 结点和关键字在每次插入后会进行调整，以保持每个叶节点等深。

以硬盘存储中使用的 B-树结构为例，逻辑结构图如下：



B-树的逻辑结构图

## 4.2. 物理结构

B-树数据结构涉及 `BTree` 类（一棵树的整体）以及 `BTreeNode` 类（树中的结点）。为方便内存管理，它们全都继承 Qt 提供的标准 `QObject` 类。它在纯 C++ 类之上，添加了父/子对象关系设置的功能以便内存管理、信号/槽的实现以便异步调用。这样，数据结构类便能继承这些优点。

`BTree` 类的定义包括 B-树的阶数 `m`，和一个 `BTreeNode *` 指针，用于指向树的哨兵结点（sentinel），该哨兵结点的唯一孩子为 B-树的根。亦即，`BTree` 对象只包括一个整数成员变量和一个指针成员变量，`BTreeNode` 对象与 `BTree` 对象分开、分散放置在内存中。

`BTreeNode` 对象存有该节点的键值数 `n`，以及键值的列表和子结点的列表。这两个列表均用 Qt 提供的 `QList<Element>` 模板类实现。这个模板类实现时，将在内存的另一处维护一个顺序表，存储键值或子节点。键值数 `n` 为整数 `qint32` 类型，键值采用可比较的 `QVariant` 类型，可以作为 `int`、`double` 等类型的任何一种。子结点则为指向其他 `BTreeNode` 的 `BTreeNode *` 指针。当该 `BTreeNode` 为叶节点时，所有的子结点皆为空（`nullptr`）。

这两个类的头文件（列出了成员变量和成员函数）：

```
class BTree : public QObject
{
    Q_OBJECT
public:
    // 显式构造函数
    explicit BTree(QObject *parent, qint32 m);
```

```

// 哨兵结点
BTreeNode *sentinel;
// 阶
qint32 m;

// 获取该树的根结点
BTreeNode *root();
const BTreeNode *croot() const;
// 树中搜索
const QVector<QVariant> search(const QVariant &x);
// 树中添加键值
bool add(const QVariant &x);
// 树中删除键值
bool del(const QVariant &x);

signals:
    // 信号函数，当树的内容更新时会释放该信号
    void updated();

public slots:

};

```

```

class BTreeNode : public QObject
{
    Q_OBJECT

private:
    // 拆分结点的算法
    static void split(BTreeNode *toBeSplitted, qint32 m);
    // 在叶结点删除一个键值
    void delAtLeaf(qint32 m, qint32 searchIndex);
    // 处理结点缺少足够的键值的算法
    static void dealWithKeywordShortage(BTreeNode *me, qint32 m, qint32
keyWordBackup);
    // 插入键值的算法
    bool insertKey(const QVariant *x, BTreeNode *childNodeLeft, BTreeNode
*childNodeRight, qint32 m, qint32 searchIndex = -1);
    // 获取父结点
    BTreeNode *parentNode();
    // 根据键值在本结点的索引来删除键值的算法

```



```

void delByIndex(qint32 m, qint32 searchIndex);

public:
    // 显式的构造函数
    explicit BTreeNode(QObject *parentNode = nullptr);
    // 结点的键值数，等于 keyWords 的长度
    qint32 n;
    // 键值表
    QList<QVariant> keyWords;
    // 子结点表
    QList<BTreeNode *> childNodes;
    // 在以该结点为根的子树中查找
    const QVector<QVariant> search(const QVariant *x);
    // 在以该结点为根的子树中添加键值
    bool add(const QVariant *x, qint32 m);
    // 在以该结点为根的子树中删除键值
    bool del(const QVariant *x, qint32 m);

signals:
public slots:

};

```

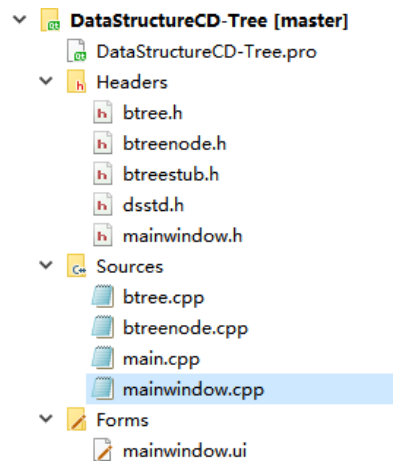
## 5. 开发平台

采用 Qt 5 5.11.1 作为开发框架，利用 Qt Creator 作为 IDE，构建套件为适用于桌面 Windows 的 MSVC2017 64 位工具链。最终构建产物为可在 Windows 上运行的.exe 应用程序。

## 6. 系统的运行结果分析说明

### 6.1. 开发、调试过程

使用 Qt Creator 开发，其文件架构如下：



调试的方法采用编写代码-运行-查看结果-启动调试器-设置断点，运行-查看运行时变量内容-调整代码-运行的流程。

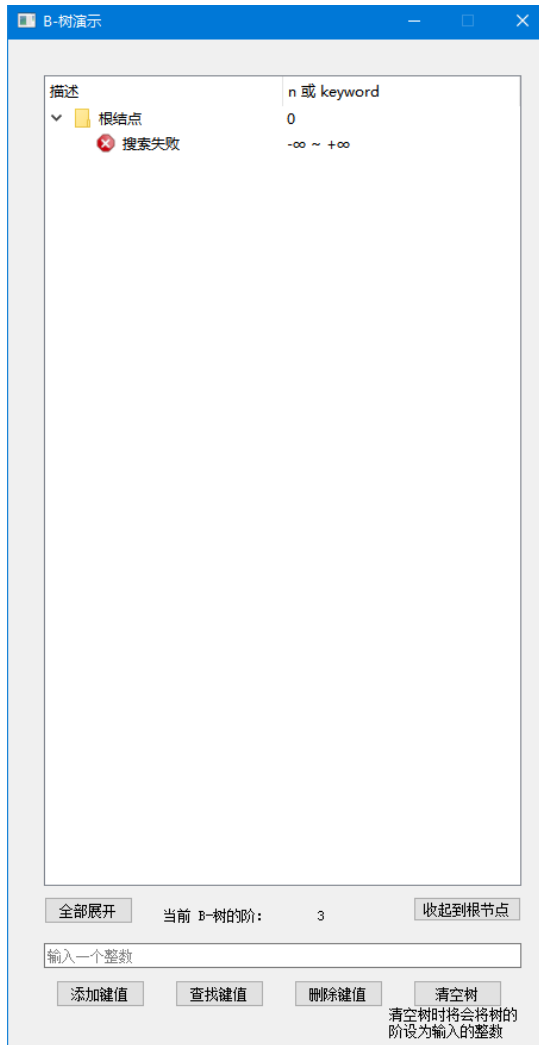
## 6.2. 开发成果

完整、无缺陷地实现了软件应有的功能。

## 6.3. 创建新树

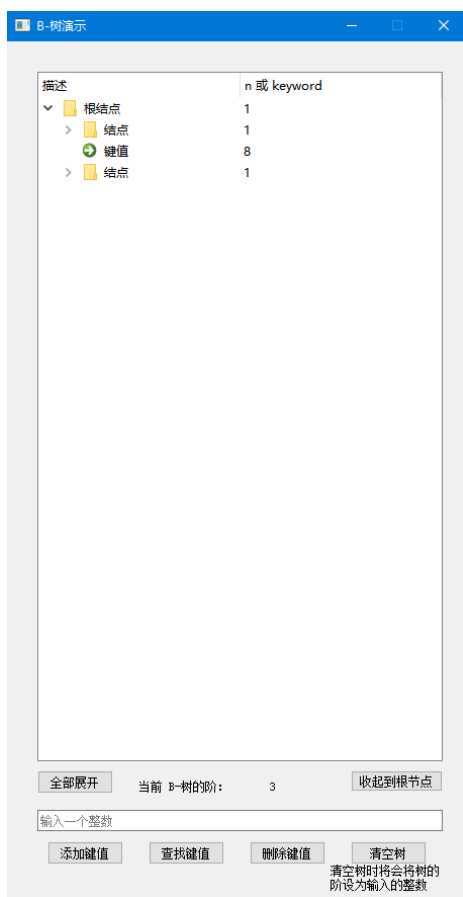
打开软件，在文本框中输入树的阶，单击“清空树”，会自动创建一棵指定阶的 B-树。软件打开时也会默认创建 3 阶 B-树。在没有任何关键字时，新的 B-树为只有空的根结点。

树的阶和结构已经在 GUI 上反馈：

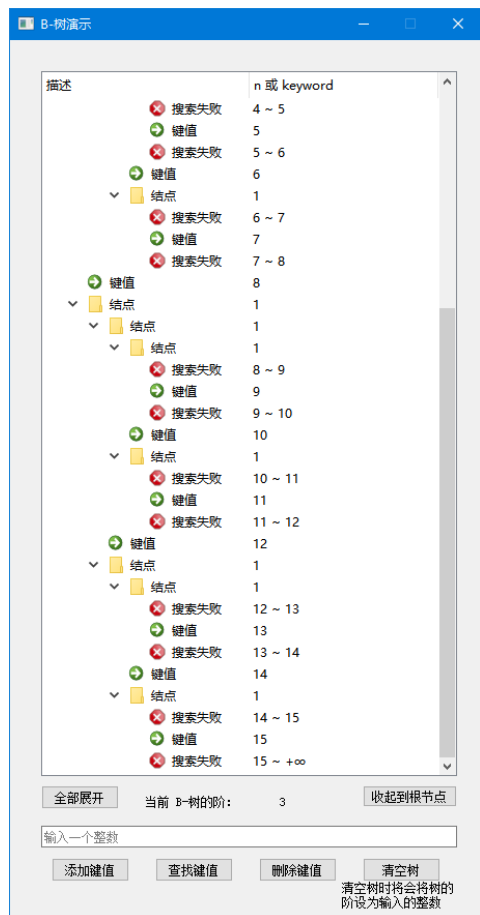


## 6.4. 添加键值

在文本框中输入整数，点击“添加键值”可以将这个整数作为键值添加到 B-树中，同时显示会更新。添加了 15 个键值后，就构造了一颗深度为 4 层的 B-树。使用“全部展开”和“收起到根节点”按钮，观察 B-树的整体和细节，如下：



观察 B-树的整体

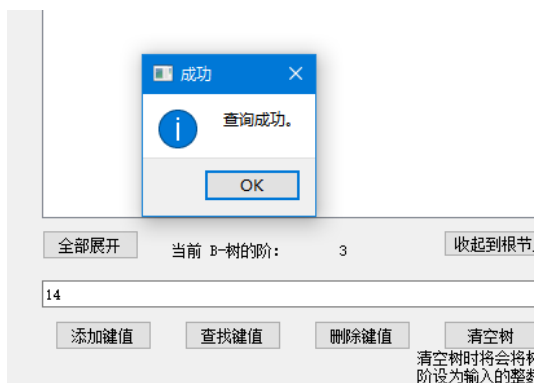


观察 B-树的细节

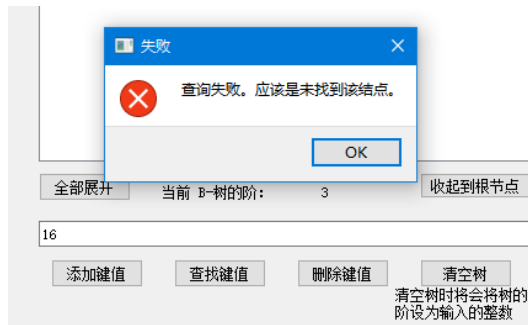
可见，这个树结构符合 3 阶 B-树的定义。

## 6.5. 查找键值

在输入框中输入要查找的键值，点击“查找键值”就会返回树的查找结果。



查询成功时



查询失败时

## 6.6. 删除键值

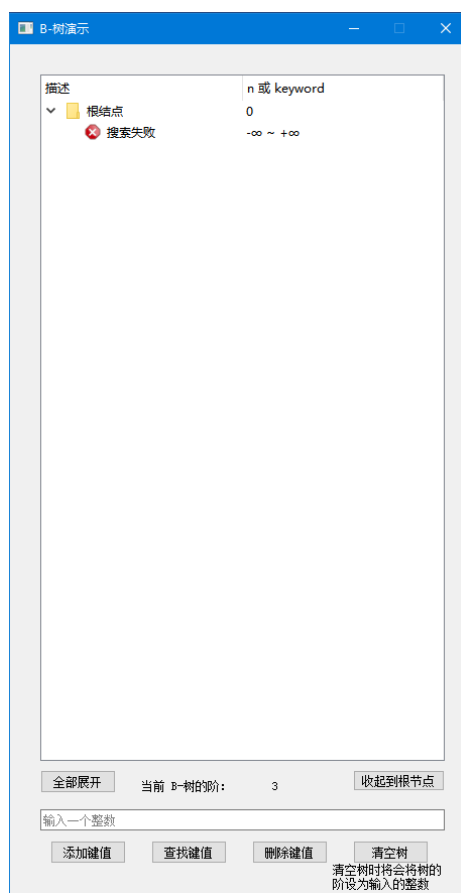
在文本框中输入树中已有的键值，点击“删除键值”按钮，将会从树中删除该键值，同时显示会更新。在删除过程中，树的结构仍然满足 B-树定义。



删除“8”和“12”后的树



删除更多结点后的树



删除所有键值，只剩空根节点的树

## 7. 操作说明

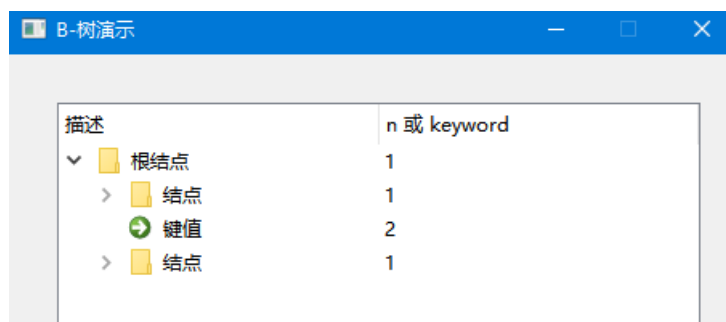
### 7.1. 基本操作说明

本部分的图片见 6。

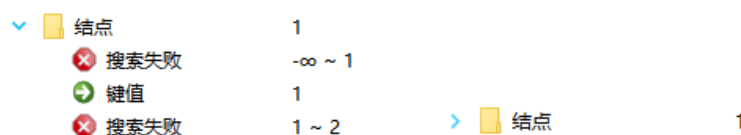
- 打开软件，在文本框中输入树的阶，单击“清空树”，会自动创建一棵指定阶的 B-树。软件打开时也会默认创建 3 阶 B-树。
- 在文本框中输入整数，点击“添加键值”可以将这个整数作为键值添加到 B-树中，同时显示会更新。
- 在输入框中输入要查找的键值，点击“查找键值”就会返回树的查找结果。
- 在文本框中输入树中已有的键值，点击“删除键值”按钮，将会从树中删除该键值，同时显示会更新。

## 7.2. 树的显示、查看

- 树的显示方式为类 Windows 下的文件目录视图。
- 点击“收起到根结点”时，树会收起到根节点，方便整理查看。



- 单击每个“文件夹”项（对应数据结构中的结点）侧边的箭头（或双击该项），可展开/收起该结点。

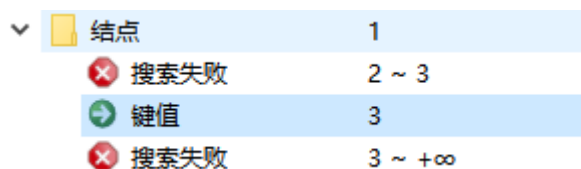


展开/收起后的结点

- “键值”项对应数据结构中结点中的键值，与同一结点的子结点并列。



- 叶结点中的键值之间会出现“查找失败”项，同时给出待查找键值所在的范围。“查找失败”项对应数据结构结点中的空指针（或失误结点），搜索到“查找失败”时，说明键值不存在于这棵树内。



- 点击“全部展开”时，所有的项全部展开，方便观察树的细节。

描述	n 或 keyword
√ 根结点	1
√ 结点	1
✖ 搜索失败	$-\infty \sim 1$
➡ 键值	1
✖ 搜索失败	$1 \sim 2$
➡ 键值	2
√ 结点	1
✖ 搜索失败	$2 \sim 3$
➡ 键值	3
✖ 搜索失败	$3 \sim +\infty$

- 软件会记住当前的状态是为“全部展开”还是“收起到根结点”，并在树更新时根据当前状态重新构造目录项。



---

## 第二部分 综合应用设计说明

### 1. 题目

（原为第 0 题，选择更难的第 2 题）

上海的地铁交通网络已基本成型，建成的地铁线十多条，站点上百个，现需建立一个换乘指南打印系统，通过输入起点站和终点站，打印出地铁换乘指南，指南内容包括起点站、换乘站、终点站。

- (1) 图形化显示地铁网络结构，能动态添加地铁线路和地铁站点。
- (2) 根据输入起点站和终点站，显示地铁换乘指南。
- (3) 通过图形界面显示乘车路径。

### 2. 软件功能

本软件实现一个地铁交通网络管理和线路查询系统。软件可以：

- 以二维结点（地铁站）、线（地铁线路）排布的方式来显示地铁网络；
- 允许用户通过拖动的方式重排结点（地铁站）；
- 自由添加、删除地铁站和地铁线路；
- 在本地存储中存储和读取地铁地图的配置和排布，以便重新打开软件时仍可继续上一次工作；
- 以 JSON 字符串的形式导入和导出地铁地图的配置和排布；
- 内置一幅上海轨道交通图，可自由加载到软件；
- 查询给定的起点站和终点站之间的最短线路，并以图形和文字的方式展示乘车路径。

### 3. 设计思想

本软件的实现方式为 HTML5 + CSS + JavaScript，主体编程语言为 JavaScript。

#### 3.1. 数据结构设计思想

对于图形化展示地铁线路、导入导出地铁线路等功能，用一套较为简单的数据结构存储地铁地图，包括站点、线路；而对于查询最短路径的功能，不能直接用前述数据结构来

---

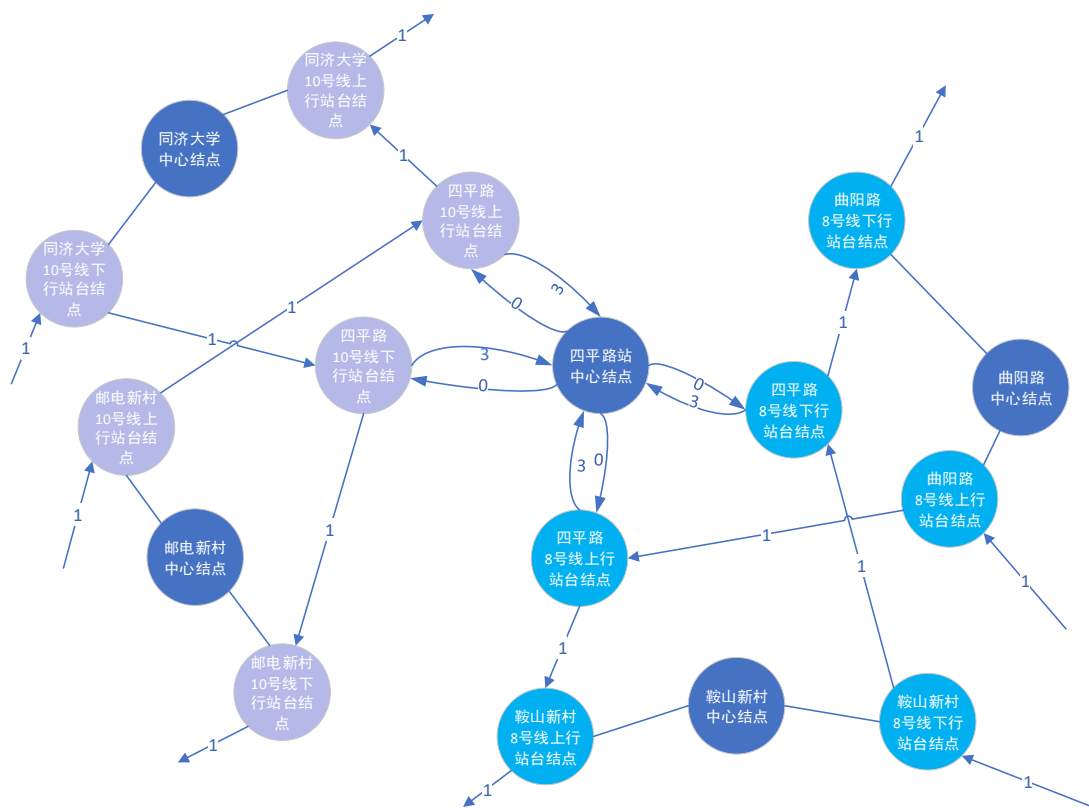
计算，而要将前述数据结构转换为有向图的形式，使用 Dijkstra 算法求得地铁最短路径。

在计算最短路径时，有这样一些约束：

- 简单地将相邻两站之间的距离认为是 1（鉴于方便而定，若有自定义每站间距离的需求，此数据结构和算法也可在稍微修改后兼容）；
- 用户会存在为不换乘而坐多几站的想法，故将地铁的同站换乘产生的距离认为是一个固定常数，换乘权重  $c$ （每次计算时可由用户指定）；
- 将支线和主线看作两条不同的地铁线，但允许将环线看作一条线。

在这样的约束下，可以将前述数据结构转换为一种特殊的有向图来方便求得最短路径。转换的函数名称为 `dataToGraph()`，其转换方法描述如下：

对于每一个站点，创建  $n$  个图上的结点，其中  $n=1+(\text{该地铁站连接到的线路数})\times 2$ 。这些结点首先包括站点的“中心结点”，和一系列“站台结点”。每条连接的地铁线路的上行方向和下行方向各有一个“站台结点”。每个站点的“中心结点”只和自己的“站台结点”连接，其“入”方向的弧权值为  $c$ ，“出”方向的弧权值为 0。站台之间的连接，用同一方向、同一线路的“站台节点”的连接实现。如所有“2 号线上行方向”的结点按运行次序用弧连接。这样，所有的结点连接在一起，可以模拟地铁的行程、换乘关系。



四平路站附近的结点关系（换乘权重为3）

建立图之后，可以对其用 Dijkstra 最短路径算法求得最短路径。源结点选为出发站点的“中心结点”，目标结点选为目的站点的“中心结点”。计算出后，由于从目的站点的“站台结点”进入“中心节点”额外计算了路径，故路径值应减去换乘权重的一倍。

由于地铁站的设计要求，每个站通过的线路不应该很大，故考虑为常数级。这样，图的结点数为站点数的常数级倍，则算法的时间复杂度为 $O(n^2)$ （ $n$ 为站点数）。

### 3.2. 图形界面设计思想

用可以在 HTML5 里直接展示的 SVG 图像格式来展示地铁地图；同时，用基于事件响应的 JavaScript 方法来处理用户输入、改变界面图像、给出输出。

---

## 4. 逻辑结构与物理结构

### 4.1. 必要的说明

本软件的数据结构用 JavaScript 实现。JavaScript 除整数、字符串等基本数据类型外，对于数组、对象的赋值都是引用传递，故物理结构在实现逻辑结构时，每个对象中，基本数据类型的属性存放在对象的内存空间内部，而子数组、子对象类型的属性存放在内存的其他位置，通过将它们的引用绑定到父对象的属性来实现联系。这样，物理结构和逻辑结构的转换较为容易，故此处略去物理结构的说明，只说明逻辑结构。

JavaScript 不是面向对象的编程语言，而是基于原型（Prototype-based）的编程语言，但可以模拟面向对象式的编程。下面将介绍两类主要数据结构的原型。

### 4.2. 地图数据结构

地图数据结构用全局变量 `mapData` 存储，其代码如下：

```
mapData = {stations: {}, lines: {}}
```

`mapData` 具有两个属性：`stations`，以车站 ID 为索引，`Station` 对象为值，存储所有的车站；`lines`，以线路 ID 为索引，`Line` 对象为值，存储所有的线路。一个实际使用中的 `mapData` 中结构如下：

```

< ▼ {stations: {...}, lines: {...}}
  ▼ lines:
    ▶ b10: Line {name: "10号线", id: "b10", drawOffset: 0, color: "#bba7d6", stations: Array(27), ...}
    ▶ b11: Line {name: "11号线", id: "b11", drawOffset: 0, color: "#7b2031", stations: Array(28), ...}
    ▶ e02: Line {name: "2号线东延线", id: "e02", drawOffset: 0, color: "#8dc600", stations: Array(9), ...}
    ▶ l01: Line {name: "1号线", id: "l01", drawOffset: 0, color: "#e52035", stations: Array(28), ...}
    ▶ l02: Line {name: "2号线", id: "l02", drawOffset: 0, color: "#8dc600", stations: Array(22), ...}
    ▶ l03: Line {name: "3号线", id: "l03", drawOffset: -4, color: "#ffd800", stations: Array(29), ...}
    ▶ l04: Line {name: "4号线", id: "l04", drawOffset: 4, color: "#4d2991", stations: Array(27), ...}
    ▶ l05: Line {name: "5号线", id: "l05", drawOffset: 0, color: "#8d54a7", stations: Array(11), ...}
    ▶ l06: Line {name: "6号线", id: "l06", drawOffset: 0, color: "#d21874", stations: Array(28), ...}
    ▶ l07: Line {name: "7号线", id: "l07", drawOffset: 0, color: "#ef7200", stations: Array(33), ...}
    ▶ l08: Line {name: "8号线", id: "l08", drawOffset: 0, color: "#219ce0", stations: Array(30), ...}
    ▶ l09: Line {name: "9号线", id: "l09", drawOffset: 0, color: "#7fc7f2", stations: Array(26), ...}
    ▶ l10: Line {name: "10号线", id: "l10", drawOffset: 0, color: "#bba7d6", stations: Array(28), ...}
    ▶ l11: Line {name: "11号线", id: "l11", drawOffset: 0, color: "#7b2031", stations: Array(31), ...}
    ▶ l12: Line {name: "12号线", id: "l12", drawOffset: 0, color: "#118840", stations: Array(32), ...}
    ▶ l13: Line {name: "13号线", id: "l13", drawOffset: 0, color: "#ff88aa", stations: Array(19), ...}
    ▶ l16: Line {name: "16号线", id: "l16", drawOffset: 0, color: "#77ccaa", stations: Array(13), ...}
    ▶ m00: Line {name: "磁悬浮线", id: "m00", drawOffset: 0, color: "#888800", stations: Array(2), ...}
    ▶ __proto__: Object
  ▼ stations:
    ▶ b10s25: Station {x: 427, y: 914, name: "龙柏新村", id: "b10s25", captionOffsetX: 10.375, ...}
    ▶ b10s26: Station {x: 398, y: 944, name: "紫藤路", id: "b10s26", captionOffsetX: -4.984375, ...}
    ▶ b10s27: Station {x: 350, y: 944, name: "航中路", id: "b10s27", captionOffsetX: 0, ...}
    ▶ b11s22: Station {x: 282, y: 388, name: "上海赛车场", id: "b11s22", captionOffsetX: -30, ...}
    ▶ b11s23: Station {x: 247, y: 415, name: "昌吉东路", id: "b11s23", captionOffsetX: 35, ...}
    ▶ b11s24: Station {x: 215, y: 447, name: "上海汽车城", id: "b11s24", captionOffsetX: 35, ...}
    ▶ b11s25: Station {x: 161, y: 464, name: "安亭", id: "b11s25", captionOffsetX: 0, ...}
    ▶ b11s26: Station {x: 111, y: 464, name: "兆丰路", id: "b11s26", captionOffsetX: 0, ...}
    ▶ b11s27: Station {x: 58, y: 464, name: "光明路", id: "b11s27", captionOffsetX: 0, ...}
    ▶ b11s28: Station {x: 4, y: 464, name: "花桥", id: "b11s28", captionOffsetX: 0, ...}
    ▶ e02s01: Station {x: 1767, y: 1280, name: "浦东国际机场", id: "e02s01", captionOffsetX: 30, ...}
    ▶ e02s02: Station {x: 1737, y: 1248, name: "海天三路", id: "e02s02", captionOffsetX: 30, ...}
    ▶ e02s03: Station {x: 1681, y: 1228, name: "远东大道", id: "e02s03", captionOffsetX: 5.796875, ...}
    ▶ e02s04: Station {x: 1625, y: 1228, name: "凌空路", id: "e02s04", captionOffsetX: -8.8046875, ...}
    ▶ e02s05: Station {x: 1562, y: 1228, name: "川沙", id: "e02s05", captionOffsetX: -35, ...}
    ▶ e02s06: Station {x: 1551, y: 1168, name: "华夏东路", id: "e02s06", captionOffsetX: 35, ...}
    ▶ e02s07: Station {x: 1527, y: 1132, name: "创新中路", id: "e02s07", captionOffsetX: 30, ...}

```

## Station 数据结构

Station 对象的原型代码如下：

```

var StationPrototype = {
  // __proto__: {
    x: 30,
    y: 30,
    name: "[UNNAMED]",
    id: undefined,
    captionOffsetX: 0,
    captionOffsetY: stationRadius,
  // }

  _nested_: undefined,
  _elemStation_: undefined,
  _elemCaption_: undefined,

  set nested(value){
    if (this._nested_ !== undefined)
      this._nested_.remove()

    this._nested_ = value
  },

  get nested(){
    return this._nested_
  },

```

```

    removeNested: function(){
        this.nested = undefined
    },

    get elemStation(){
        return this._elemStation_
    },

    get elemCaption(){
        return this._elemCaption_
    },

    draw: function() {},

    drawCaption: function() {},

    dispose: function() {}
}

```

除去一些与绘制有关的属性外，其基本属性是：

- X, y 坐标；
- 名称 (name)；
- ID；
- 站台文字相对于站台图形元素来说的偏移位置 (captionOffsetX, captionOffsetY)。

## Line 数据结构

Line 对象的原型代码如下：

```

var LinePrototype = {
    // __proto__: {
    //     name: "[UNNAMED]",
    //     id: undefined,
    //     drawOffset: 0,
    //     color: "red",
    //     stations: [],
    // }
    _groupLine_: undefined,
    _elemLegend_: undefined,
    svgLines: [],

    set groupLine(value)
    {
        if (this._groupLine_ != undefined){
            this._groupLine_.remove()
        }

        this._groupLine_ = value
    },

    get groupLine(){return this._groupLine_},

    drawLine: function(x1, y1, x2, y2){},

    draw: function() {},

```

```
updateAtStation: function(stationID){},  
dispose: function(){}  
}
```

除去一些与绘制有关的属性外，其基本属性是：

- 名称（name）；
- ID；
- 颜色（color）；
- 站点 ID 列表（stations）。这些站点 ID 可以与 mapData.stations 中的站点建立联系。

### 4.3. 有向图数据结构

用于 Dijkstra 算法的有向图以邻接表的形式存储在全局对象 mapGraph 中。mapGraph 的属性除一个记录结点数量的属性 nodeNum 外，都用结点 ID 作为索引，存储所有的结点。一个实际使用中的 mapGraph 对象结构如下：

```
> mapGraph  
< {nodeNum: 1148, l01s01: {...}, l01s02: {...}, l01s03: {...}, l01s04: {...}, ...}   
  ▼ b10s25:  
    ▼ adj: Array(2)  
      ▶ 0: (2) ["b10s25_b10_pos", 0]  
      ▶ 1: (2) ["b10s25_b10_neg", 0]  
      length: 2  
      ▶ __proto__: Array(0)  
    ▶ __proto__: Object  
  ▼ b10s25_b10_neg:  
    ▼ adj: Array(2)  
      ▶ 0: (2) ["b10s25", 3]  
      ▶ 1: (2) ["l10s05_b10_neg", 1]  
      length: 2  
      ▶ __proto__: Array(0)  
    indexInline: 24  
    ▶ __proto__: Object  
  ▶ b10s25_b10_pos: {indexInline: 24, adj: Array(2)}  
  ▶ b10s26: {adj: Array(2)}  
  ▶ b10s26_b10_neg: {indexInline: 25, adj: Array(2)}  
  ▶ b10s26_b10_pos: {indexInline: 25, adj: Array(2)}  
  ▶ b10s27: {adj: Array(2)}  
  ▶ b10s27_b10_neg: {indexInline: 26, adj: Array(2)}  
  ▶ b10s27_b10_pos: {indexInline: 26, adj: Array(1)}  
  ▶ b11s22: {adj: Array(2)}  
  ▶ b11s22_b11_neg: {indexInline: 21, adj: Array(2)}  
  ▶ b11s22_b11_pos: {indexInline: 21, adj: Array(2)}
```

其中的结点 ID 以 “\_” 号划分为三段的，为“站台结点”，三段字符串分别代表车站 ID、线路 ID、正负方向；无 “\_” 号的为“中心结点”，其 ID 与车站 ID 一致。

## 5. 开发平台

使用 Visual Studio Code 作为代码编辑器，软件的成果是 HTML 文件 + JavaScript 脚本，在最新版本的 Chrome（或其他支持 HTML5、SVG 的新型浏览器）运行。

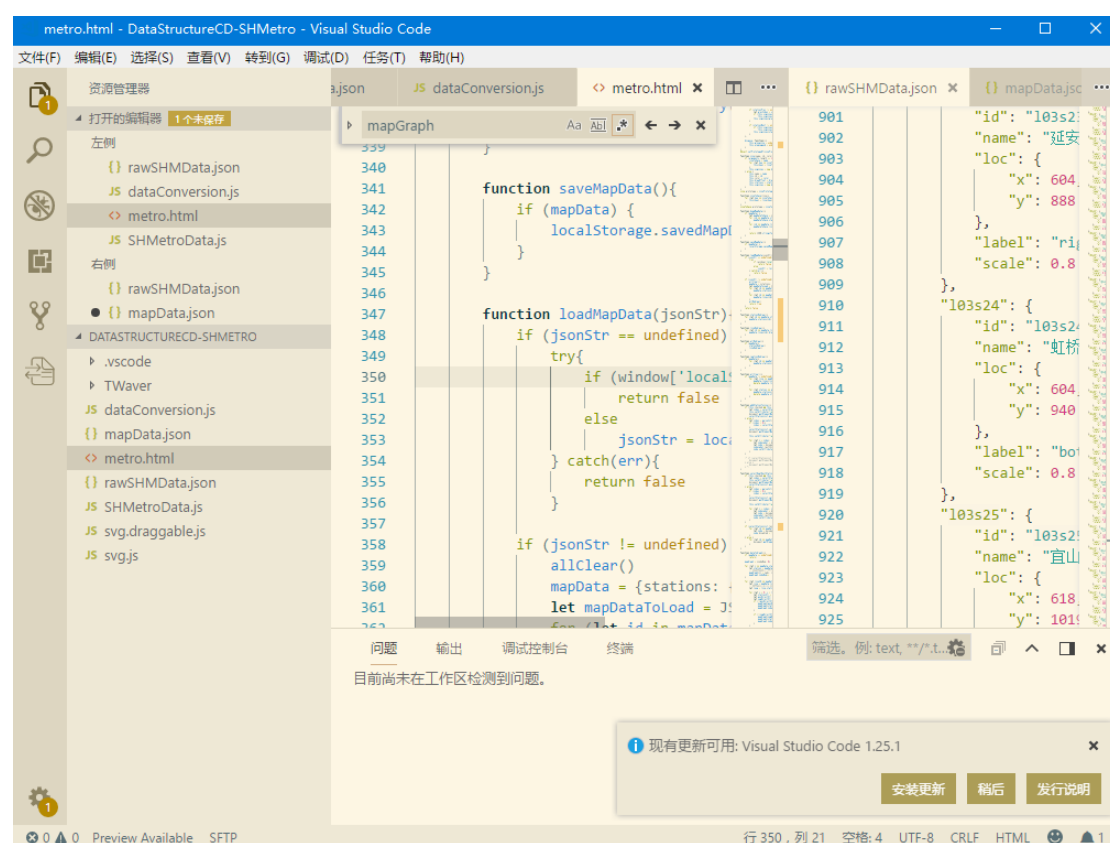
Edge 浏览在本地运行本软件时无法访问本地存储，可访问

<https://www.shunwww.pw/metro/metro.html> 使用。

## 6. 系统的运行结果分析说明

### 6.1. 开发及调试过程

本软件的代码编写在 Visual Studio Code 中完成：



在开发的过程中，用到了开源代码：SVG.js 和 SVG.draggable.js，以实现 JavaScript 对 SVG 元素更为便捷的操作。

调试过程为：编写代码-保存-浏览器运行-观察结果-调整代码。

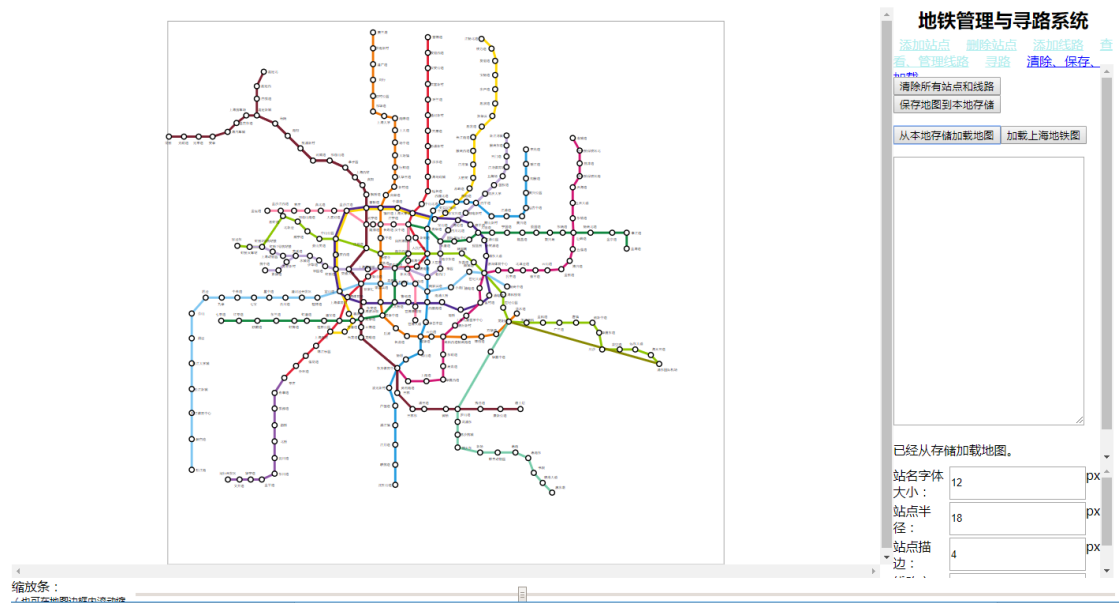
### 6.2. 开发成果

开发成果完整实现了要求的功能。

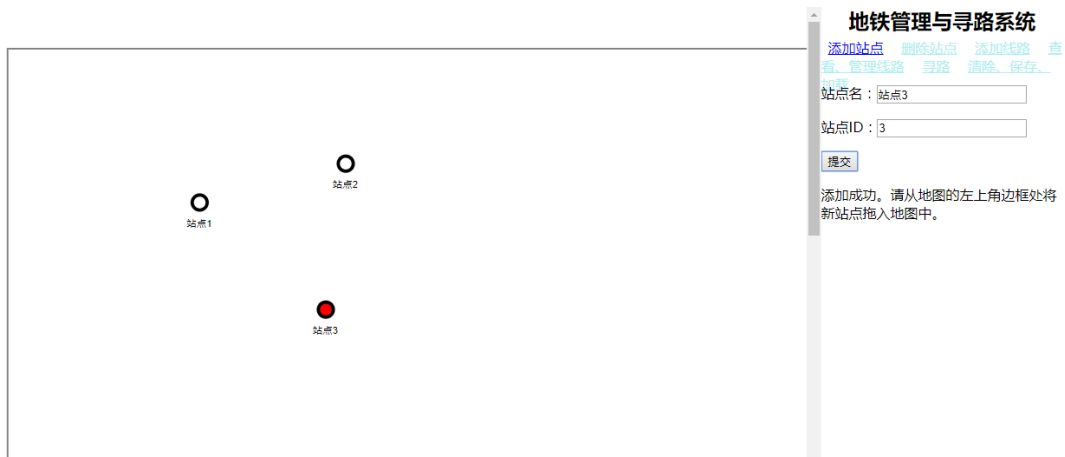


### 6.3. 运行结果

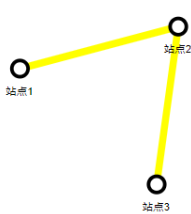
二维显示地铁图（加载内置上海地铁图并显示）



在空地图上添加站点



# 添加线路



站点1

站点2

站点3

地铁管理引导系统

[添加站点](#) [删除站点](#) [添加线路](#) [查看](#)

[管理线路](#) [寻路](#) [清除](#) [保存](#)

线路ID:

线路绘制偏移量(默认为0):

线路颜色:

请按顺时针方向, 按顺序选择该线路的途经站点:


☐ 1: 站点1

☐ 2: 站点2

☐ 3: 站点3

☐ 环线

# 删除线路



站点1

站点2

站点3

地铁管理引导系统

[添加站点](#) [删除站点](#) [添加线路](#) [查看](#)

[管理线路](#) [寻路](#) [清除](#) [保存](#)

若要删除线路, 请在相应线路前的框内打勾并提交。

当前没有地铁线路。请添加一条线路。

删除成功。

# 删除站点

添加站点

删除站点

添加线路

查看

管理线路

寻路

清除

保存

请选择要删除的站点：


注意：删除站点将删除所有与站点连接的线路！

☐ 2 : 站点2

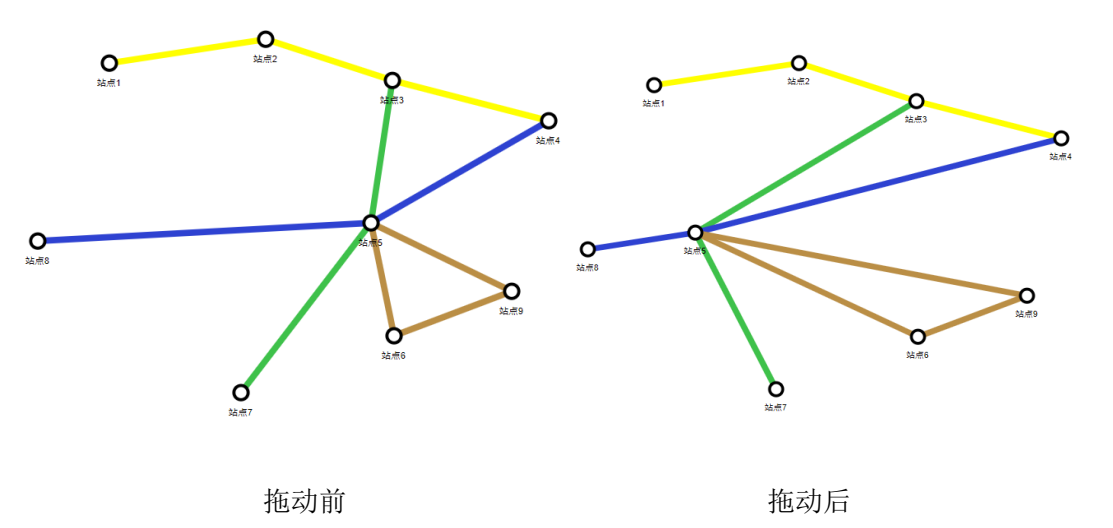
☐ 3 : 站点3

提交

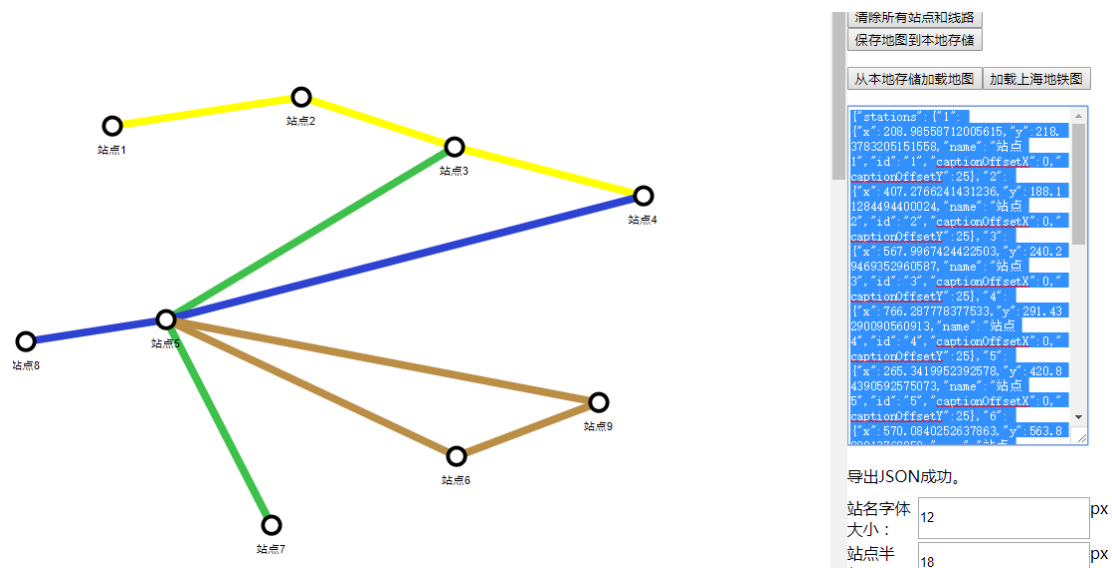
删除成功。



# 拖动站点



## JSON 导出地铁图



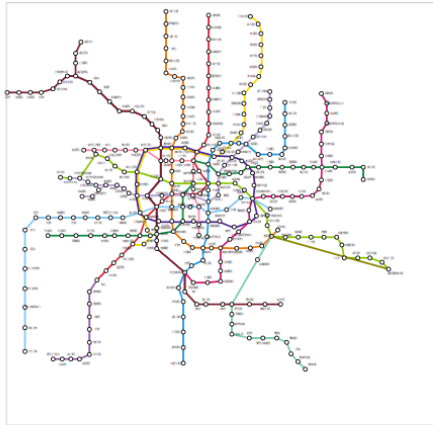
## 上海地铁寻路



## 7. 操作说明

### 7.1. 地图浏览

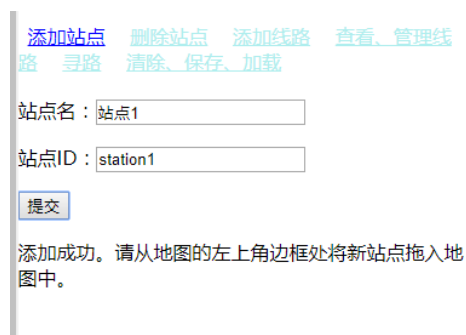
地图视图占据了网页的绝大部分面积。在地图视图上滚轮，或拖动底部的缩放条，可以进行缩放。拖动边框内的地图视图部分，可以移动视野。



## 7.2. 添加站点

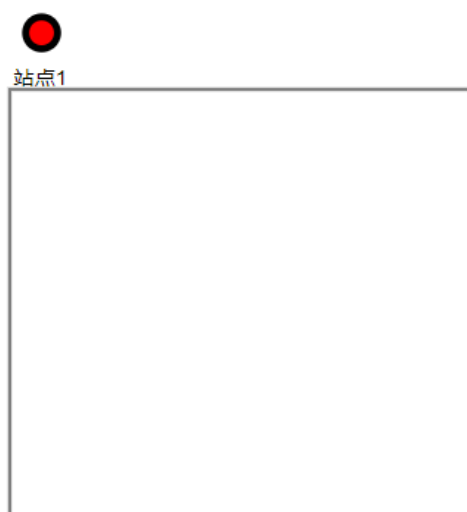
点击右侧工具栏的“添加站点”链接，显示相应的面板。填入站点名称和唯一的标识

符（ID），点击提交。



The screenshot shows a web interface for adding a station. At the top, there are several links: '添加站点' (Add Station), '删除站点' (Delete Station), '添加线路' (Add Line), '查看、管理线路' (View, Manage Line), '寻路' (Routing), and '清除、保存、加载' (Clear, Save, Load). Below these links, there are two input fields: '站点名:' (Station Name) with the value '站点1' (Station 1), and '站点ID:' (Station ID) with the value 'station1'. A '提交' (Submit) button is located below the input fields. At the bottom, a message states: '添加成功。请从地图的左上角边框处将新站点拖入地图中。' (Added successfully. Please drag the new station from the top-left corner of the map frame into the map).

接着，在地图视图的边框左上角将站点拖入。添加完成。



### 7.3. 添加线路

点击右侧工具栏的“添加线路”链接，显示相应的面板。填入线路名称和唯一的标识符（ID），选择一个颜色，并按顺序勾选线路的运行站点，选择该线是否为环线。点击提交。线路即创建完成。

[添加站点](#) [删除站点](#) [添加线路](#) [查看、管理线路](#)  
[寻路](#) [清除](#) [保存](#) [加载](#)

线路名：

线路ID：

线路绘制偏移量(默认为0)：

线路颜色：

请按顺时针方向，按顺序选择该线路的途径站点：  
☒ station1：站点1 (1)  
☒ station2：站点2 (2)

☐ 环线



## 7.4. 查看线路图例、移除线路

点击右侧工具栏的“查看、管理线路”链接，显示相应的面板。此时可查看当前添加的线路，也可勾选需要移除的线路后点击提交即可移除。

[添加站点](#) [删除站点](#) [添加线路](#) [查看、管理线路](#)  
[寻路](#) [清除](#) [保存](#) [加载](#)

若要删除线路，请在相应线路前的框内打勾并提交。

☒ ☐ l1：Line 1



## 7.5. 清除、保存、加载

点击右侧工具栏的“清除、保存、加载”链接，显示相应的面板。点击某个按钮，即可执行相应指令。

[添加站点](#)[删除站点](#)[添加线路](#)[查看、管理线路](#)[寻路](#)[清除、保存、加载](#)

清除所有站点和线路

保存地图到本地存储

从本地存储加载地图

加载上海地铁图

导出到 JSON

导入 JSON 地图数据

## 7.6. 寻路

点击右侧工具栏的“寻路”链接，显示相应的面板。按顺序选择起点站、终点站，输入换成权重  $c$ ，按提交即可显示结果。如有路径，会在地图上加粗显示。

[添加站点](#)[删除站点](#)[添加线路](#)[查看、管理线路](#)[寻路](#)[清除、保存、加载](#)

请按顺序选择起点和终点

☐ station1 : 站点1

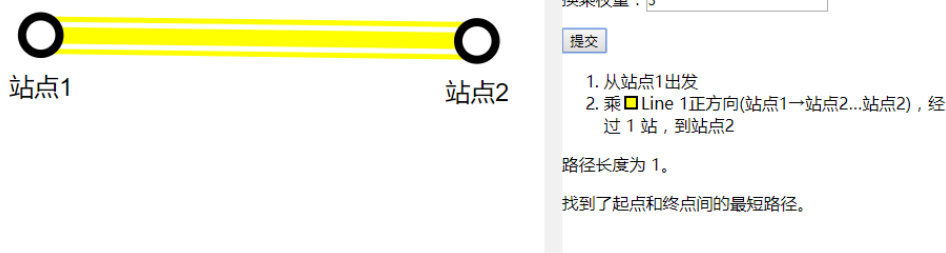
☐ station2 : 站点2

换乘权重：

提交

起点到终点之间没有路径，请检查地铁线路布置的完备性。





## 第三部分 实践总结

### 1. 所做的工作

这一个星期，在完成课设作业时我做了不少事情，比如学习 Qt 库的用法、Qt Creator 的用法、JavaScript、HTML、SVG 的相关语法等，反而实际上用在设计数据结构上的时间并不多。

### 2. 总结与收获

本作业的代码量可观，感觉学会了 Qt 和 HTML 这两类有效的界面实现工具，能够将设计的数据结构很好地展现出来，总的来说还是收获颇丰。在写代码的过程中，我认识到了设计好类与接口的重要性，以及代码整洁美观的重要性。

## 第四部分 参考文献

- [1] 维基百科.B 树[OL]:(2018-4-5)[2018-8-7].<https://zh.wikipedia.org/wiki/B%E6%A0%91>
- [2] Qt 社区.Qt Wiki[OL]:[2018-8-7]. <https://wiki.qt.io/Main>
- [3] MDN.JavaScript 指南[OL]:(2018-7-1)[2018-8-7]. <https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Guide>
- [4] Wout Fierens.SVG dot js Documentation[OL]:(2018-2-24)[2018-8-7]. <http://svgjs.com/>
- [5] 蘑菇先生. 算法数据结构(一)-B 树[OL]:(2014-11-15)[2018-8-7]. <https://www.cnblogs.com/mushroom/p/4100087.html>