**第一部分 算法实现设计说明**

* 1. **题目**

题号0：试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-树随时显示输出出阿里，能给出查找是否成功的有关信息。

* 1. **软件功能**
     1. **B树的定义**

1. B树的定义

B树也称B-树,它是一颗多路平衡查找树。我们描述一颗B树时需要指定它的阶数，阶数表示了一个结点最多有多少个孩子结点，一般用字母m表示阶数。当m取2时，就是我们常见的二叉搜索树。

一颗m阶的B树定义如下：

1）每个结点最多有m-1个关键字。

2）根结点最少可以只有1个关键字。

3）非根结点至少有Math.ceil(m/2)-1个关键字。

4）每个结点中的关键字都按照从小到大的顺序排列，每个关键字的左子树中的所有关键字都小于它，而右子树中的所有关键字都大于它。

5）所有叶子结点都位于同一层，或者说根结点到每个叶子结点的长度都相同。

**1.2.2 B树的查找**

查找操作是指查找一条记录，即（key, value）的键值对。如果B树中存在需要查找的键值对，则需要返回查找的结点以及查找键值的位置信息。若B树不存在这个key,则返回查找不成功的信息。

1. 从根节点开始，在当前结点中查找key值，若存在则返回查找的相关信息，如果不存在就在相应的子树中查找。
2. 若果相应的子树为NULL则查找失败

**1.2.3 B树的插入**

插入操作是指插入一条记录，即（key, value）的键值对。如果B树中已存在需要插入的键值对，则用需要插入的value替换旧的value。若B树不存在这个key,则一定是在叶子结点中进行插入操作。

1）根据要插入的key的值，找到叶子结点并插入。

2）判断当前结点key的个数是否小于等于m-1，若满足则结束，否则进行第3步。

3）以结点中间的key为中心分裂成左右两部分，然后将这个中间的key插入到父结点中，这个key的左子树指向分裂后的左半部分，这个key的右子支指向分裂后的右半部分，然后将当前结点指向父结点，继续进行第3步。

**1.2.4 B树的删除：**

删除操作是指，根据key删除记录，如果B树中的记录中不存对应key的记录，则删除失败。

1）如果当前需要删除的key位于非叶子结点上，则用后继key（这里的后继key均指后继记录的意思）覆盖要删除的key，然后在后继key所在的子支中删除该后继key。此时后继key一定位于叶子结点上，这个过程和二叉搜索树删除结点的方式类似。删除这个记录后执行第2步

2）该结点key个数大于等于Math.ceil(m/2)-1，结束删除操作，否则执行第3步。

3）如果兄弟结点key个数大于Math.ceil(m/2)-1，则父结点中的key下移到该结点，兄弟结点中的一个key上移，删除操作结束。

否则，将父结点中的key下移与当前结点及它的兄弟结点中的key合并，形成一个新的结点。原父结点中的key的两个孩子指针就变成了一个孩子指针，指向这个新结点。然后当前结点的指针指向父结点，重复上第2步。

有些结点它可能即有左兄弟，又有右兄弟，那么我们任意选择一个兄弟结点进行操作即可。

* 1. **设计思想**

1学习相应知识，做好必要的准备工作

由于以前都是采用控制台进行编程，即便涉及一些简单的图像界面，但是比较粗制简陋，无法入眼，并不是标准规范、人性化的用户交互界面，所以要完成本次的数据结构课程设计必须从零起步，学习可视化编程开发。在C++的一系列可视化开发框架下，我选择用Qt来实现程序的功能，因为Qt相对较为简单，容易上手入门，同时Qt是较为新兴的技术框架，并且跨平台开发，很有前景和实用性。通过几天的学习，理解掌握的Qt的必要知识，包括最为核心的信号和槽函数机制、UI控件的使用、Scene-View视图框架等核心技术。

2自顶向下设计

有了必要的准备知识，就可以进行程序的总体规划设计了。自顶向下分析是常用的分析方法，本次题目其实较为简单，此次实现图形化界面的演示需要结合原有结构，融入图形化元素和用户界面接口，对程序的功能分析，显然程序的功能分为三个子功能模块，分别对每个模块进行设计即可完成整个任务。

3自底向上实现

具体实现时，先定义每个类的属性和相应函数，然后根据定义，设计相应算法自底向上进行实现，逐个击破，最终完成所有程序的设计。

* 1. **逻辑结构与物理结构**

class BTNode {

public:

int keynum;//关键字数量

BTNode \*parent;//双亲结点

KeyType key[M + 1];//关键字数组

BTNode \*ptr[M + 1];//子树数组

BTNode();

BTNode\* getRightBrother();

BTNode\* getLeftBrother();

int getPointerPos();

Status replace(KeyType Key,KeyType newKey);

Status Erase(KeyType K);

Status eraseKey(KeyType K);

Status erasePtr(BTNode\* p);

void updateChild();

int Search(KeyType x);

Status Insert(int i, KeyType k, BTNode \*aq);

Status insertKey(int i, KeyType k);

Status insertChild(int i, BTNode \*aq);

Status Remove(int i);

bool isLeaf();

//qt

QRectF border;

QLineF line;

};

class Btree {

protected:

BTNode \*root;//根节点

public:

Btree();//构造函数

~Btree();//删除所有节点

Result\* SearchBTree(KeyType K);//查找关键字

Status InsertBTree(KeyType K);//插入关键字

Status RemoveBTree(KeyType K);//删除关键字

BTNode\* FindNextNode(BTNode\* q,KeyType K,KeyType& nextKey);

void maintainAfterErase(BTNode \*p);

void mergeNode(BTNode\*p1, BTNode\*p2, KeyType father\_key);

void DisplayBTree();

void Display(BTNode \*q,int level);

void Delete(BTNode \*q);//删除子树

BTNode\* getRoot();

int getLevel();//得到b树的高度

//qt

QVector<BTNode\*>v;

void Painting(int w, int h);

void dfs(BTNode \*p, int dep,qreal hd,qreal line\_d);

};

class Result {

protected:

public:

BTNode \*pt;//指向找到的结点

int i;//1...m在结点中关键字序号

int tag;//1：查找成功 0：查找失败

Result(BTNode \*pt, int i, int tag);

};//B-数的查找结果类型

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

Btree tree;//B树

QRectF demo\_rec;//显示B树的面板

~*MainWindow*();

protected:

void *paintEvent*(QPaintEvent\*);

private:

int change=0;

Ui::MainWindow \*ui;

void updateRec();

QPointF S;

void initSignalSlots();

void initUI();

private slots:

void on\_pushButton\_clicked();

void on\_pushButton\_2\_clicked();

void on\_pushButton\_3\_clicked();

};

* 1. **开发平台**

开发语言 ：C++（C++11标准以上）

开发框架 ：QT

集成开发环境：Qt Version 5.9.0

编译器 ：MinGW 32bit

* 1. **系统的运行结果分析说明**

**1.6.1 调试及开发过程**

1. 调试

算法开发采用的是新技术框架 Qt，同时也是跨平台的，在 Visual Studio + Qt designer中开发调试，Qt中包含了大量的库类，类似于 java 开发简便，Qt 有较好的调试器

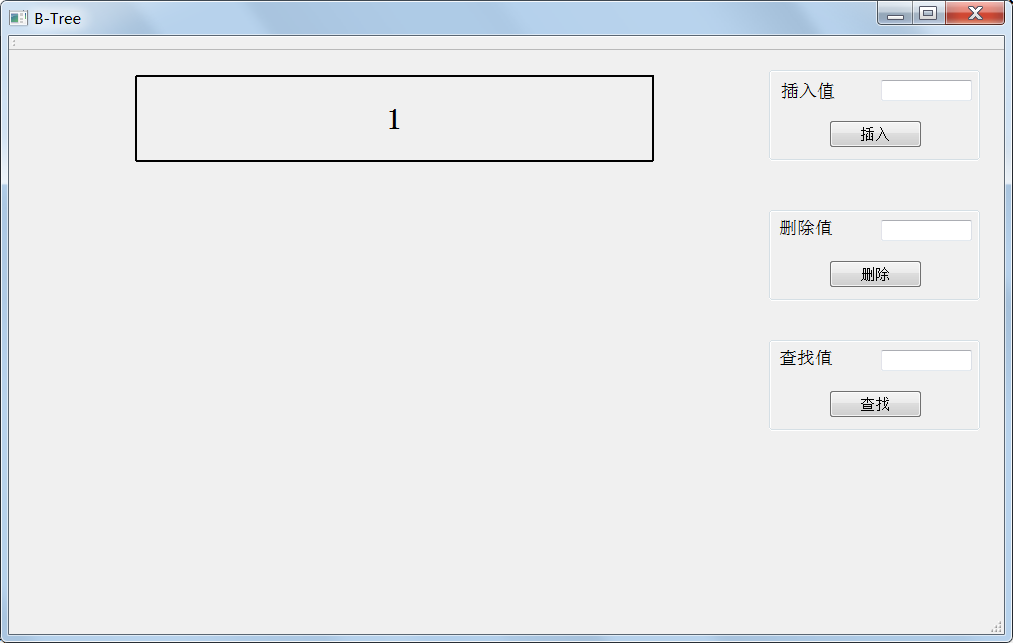
2. 开发

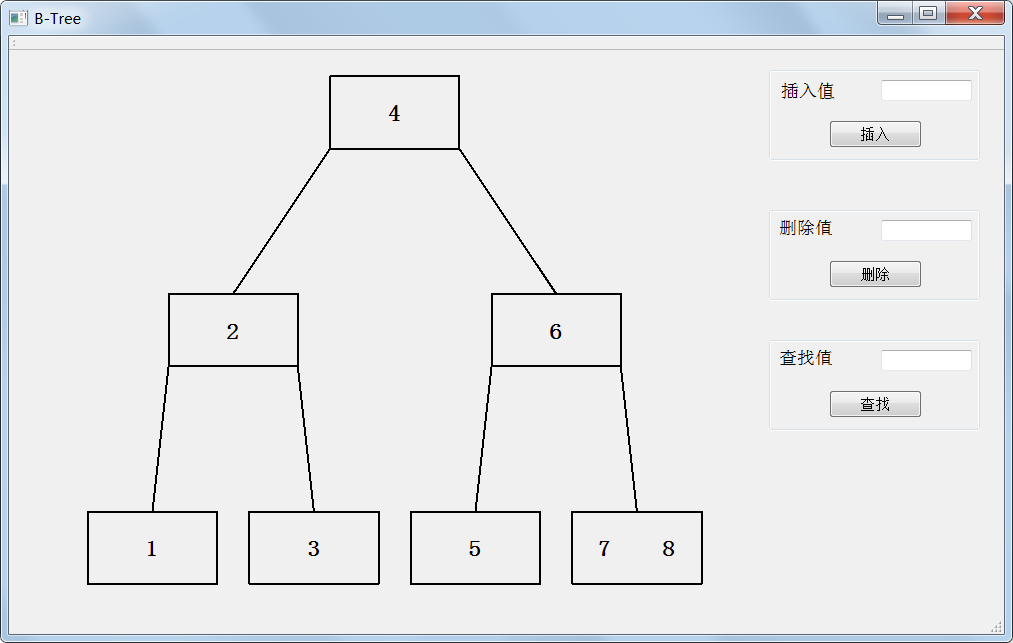
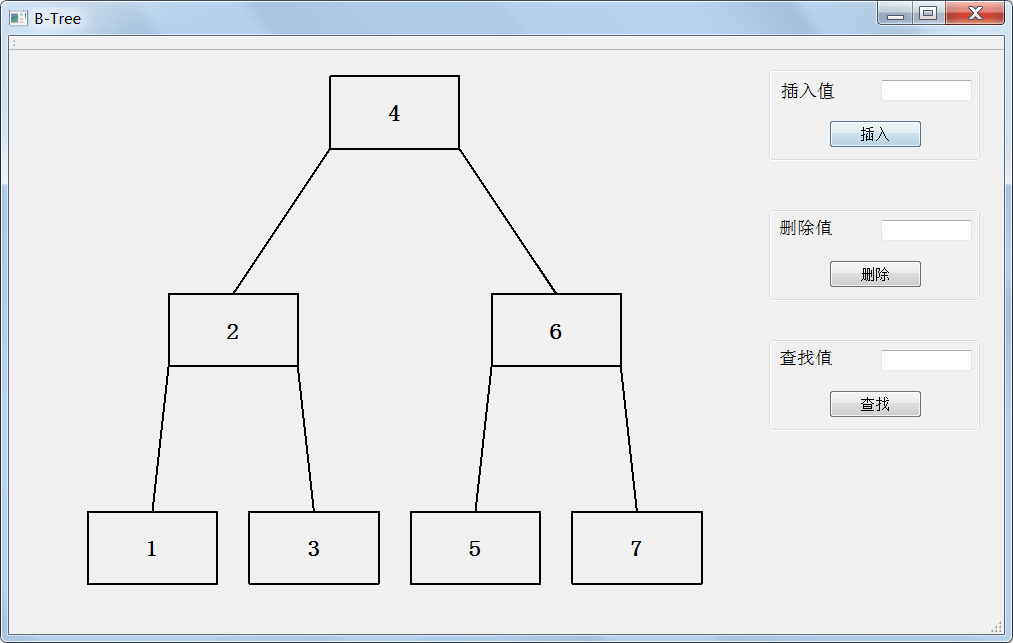
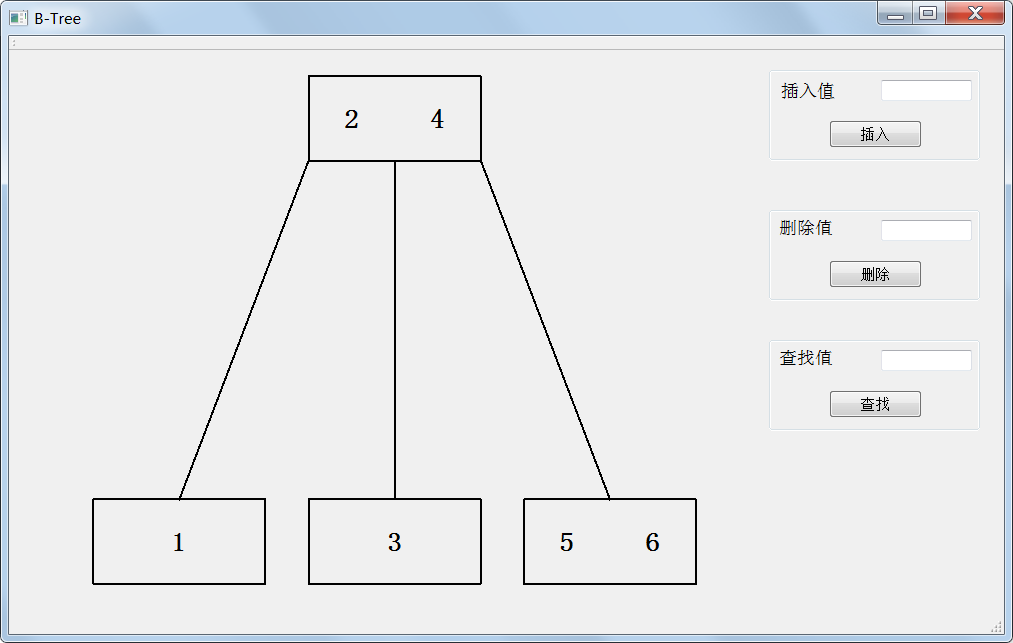
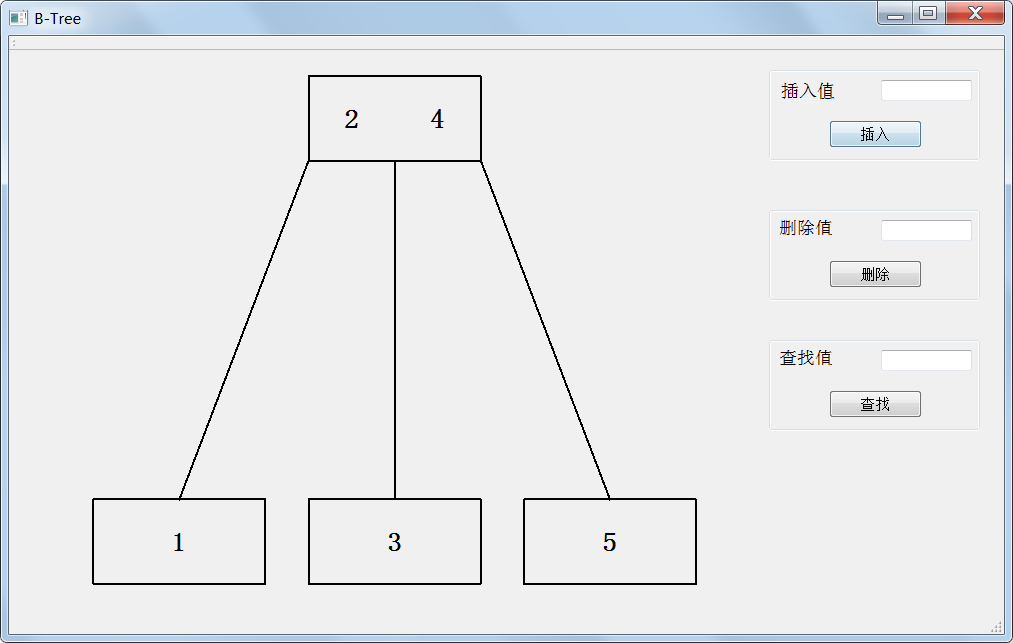
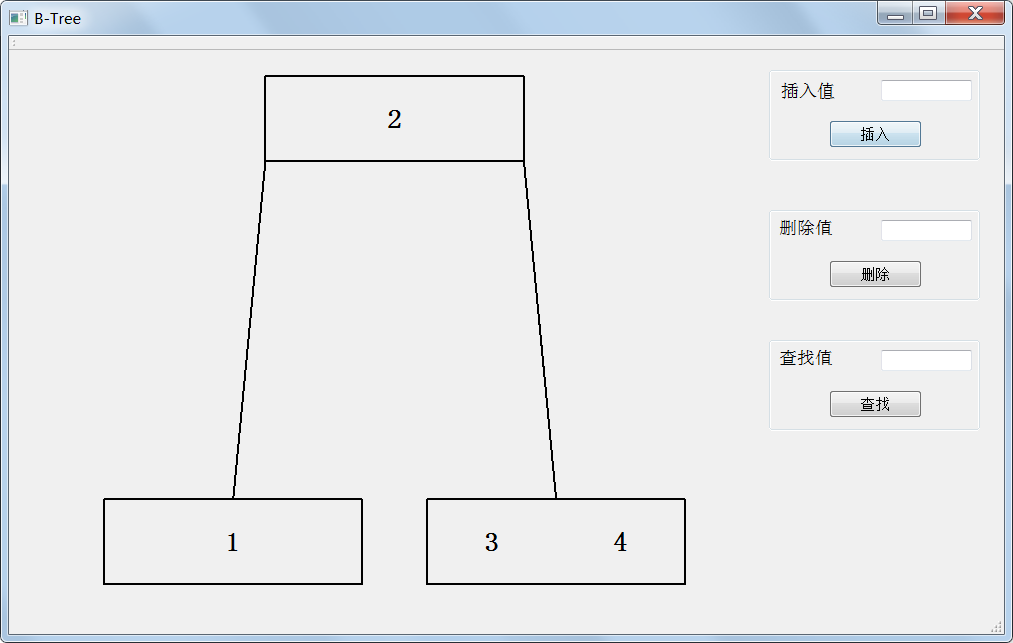
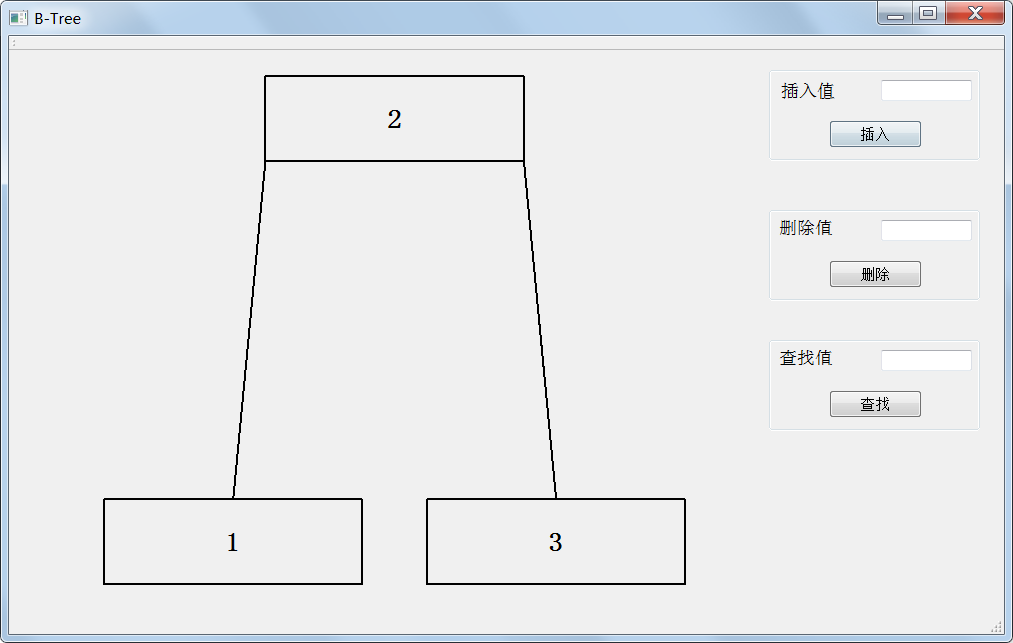
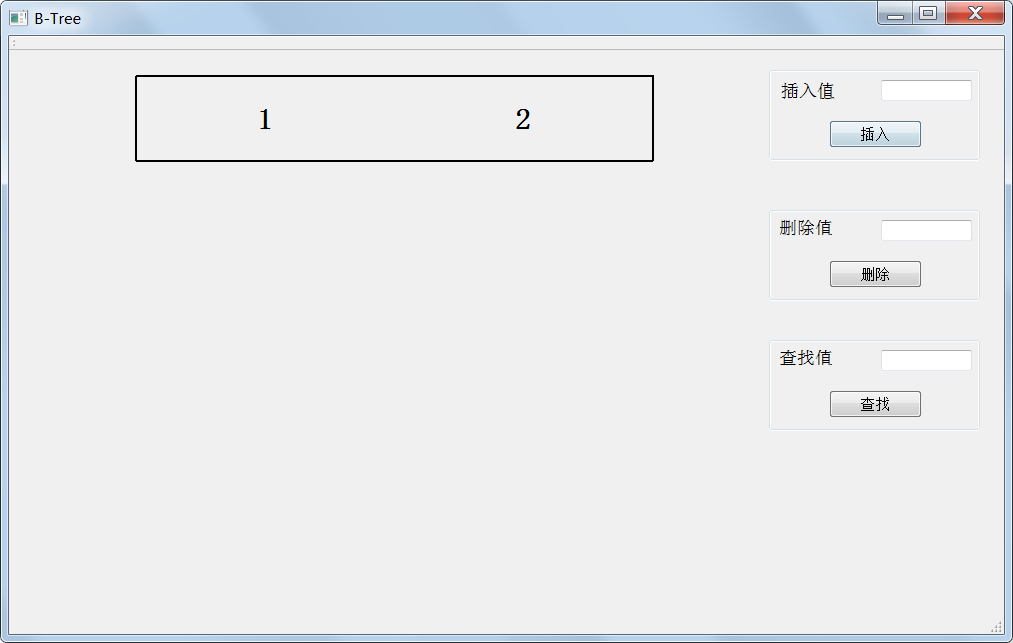
算法有三个小题，分别是在B-树中进行插入、删除操作和显示操作结果，采用分模块逐步开发方式。先确定主窗口，控制整体界面，然后完成B-树设计和开发，B-树完成以后集中完成界面显示。

**1.6.2 成果分析**

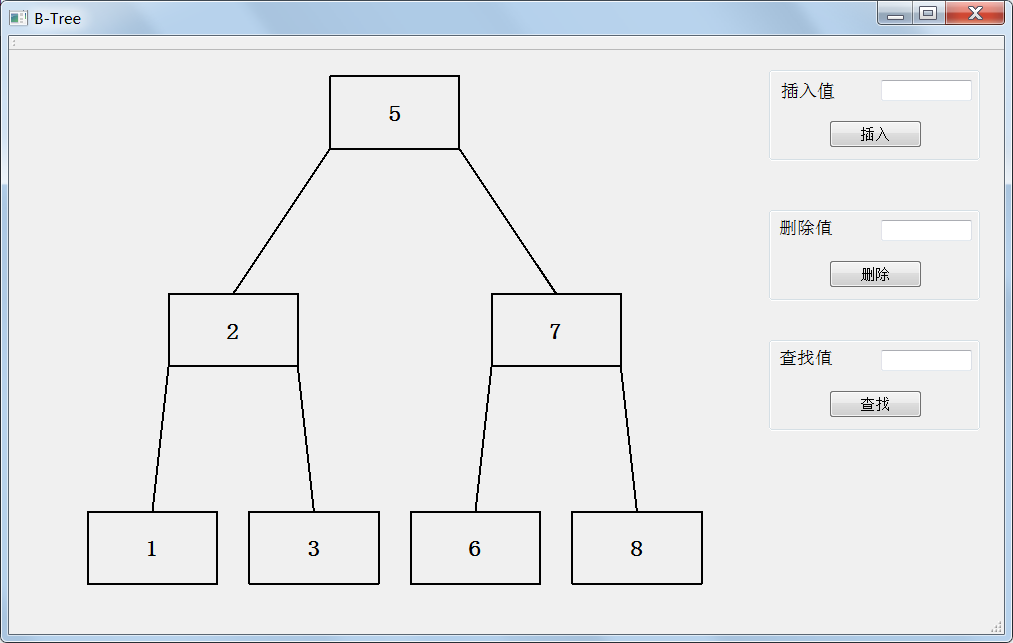
**1.6.2.1 正确性**

向B-树中依次插入1-8

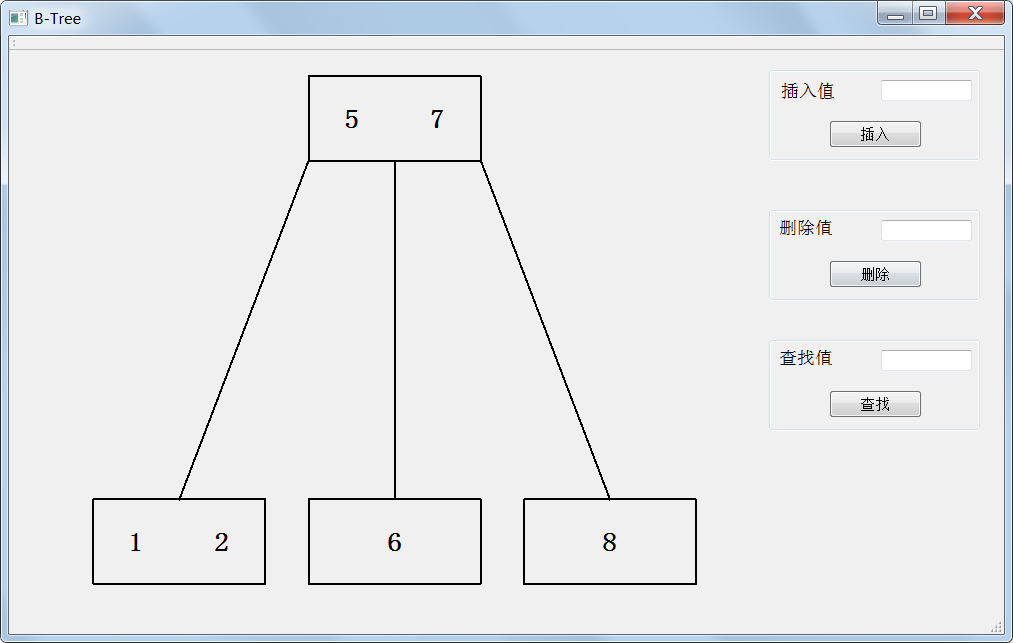




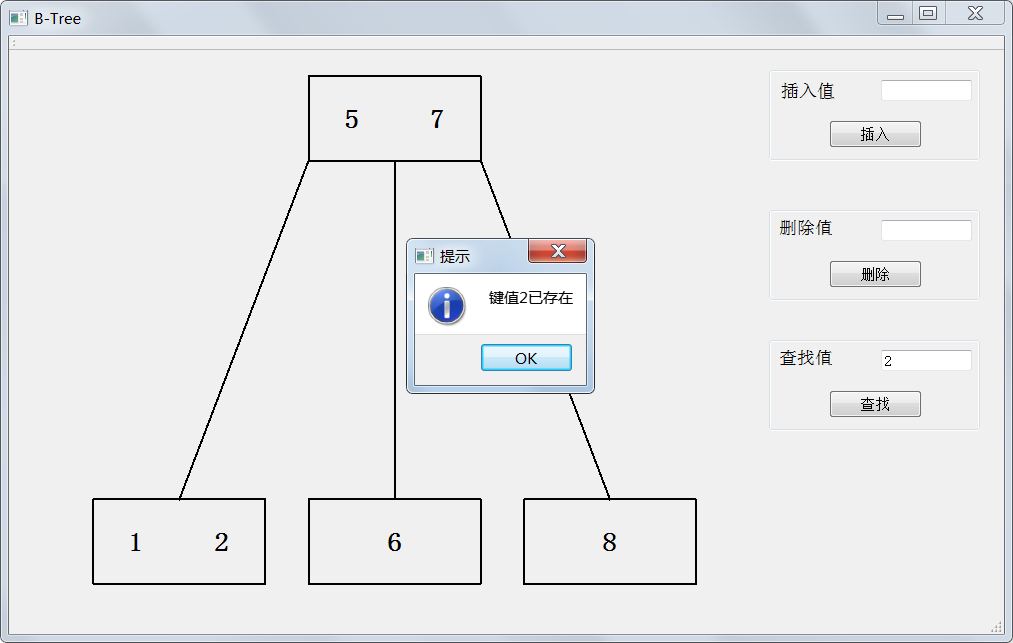
删除4：



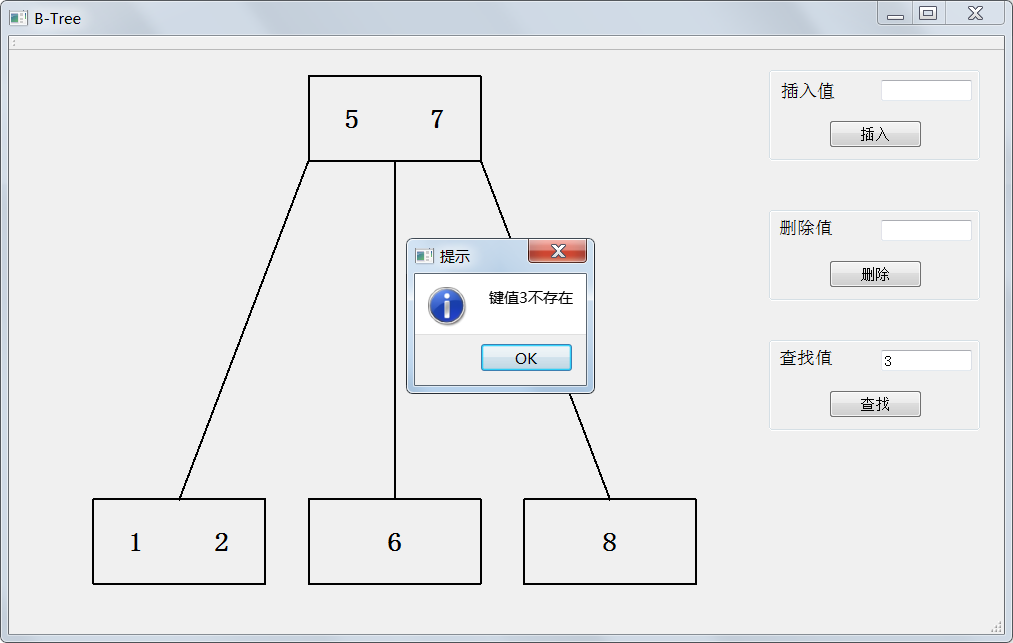
删除3：



查找2：

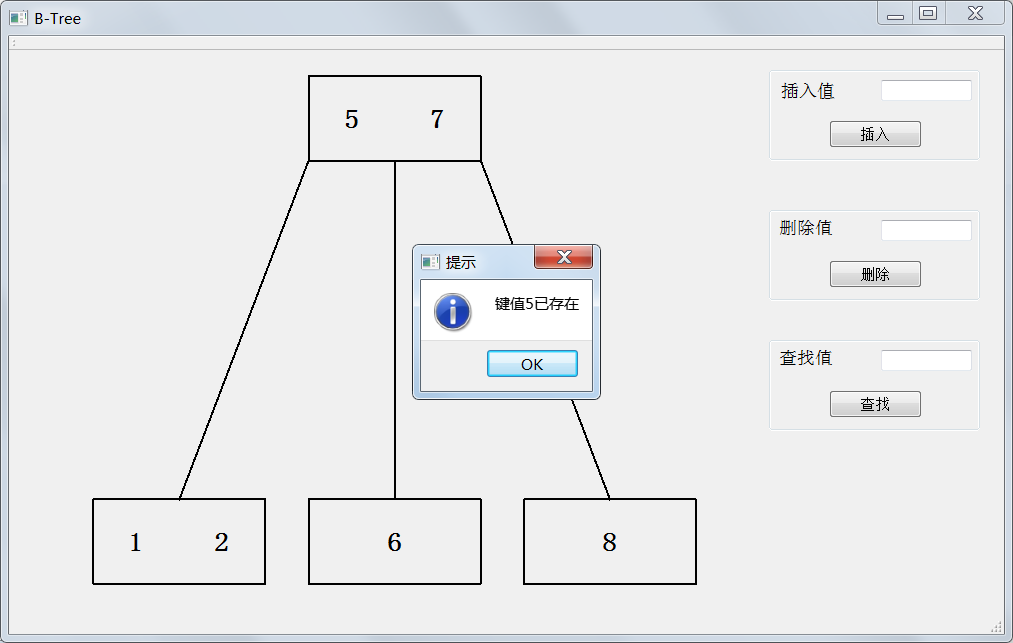


查找3：

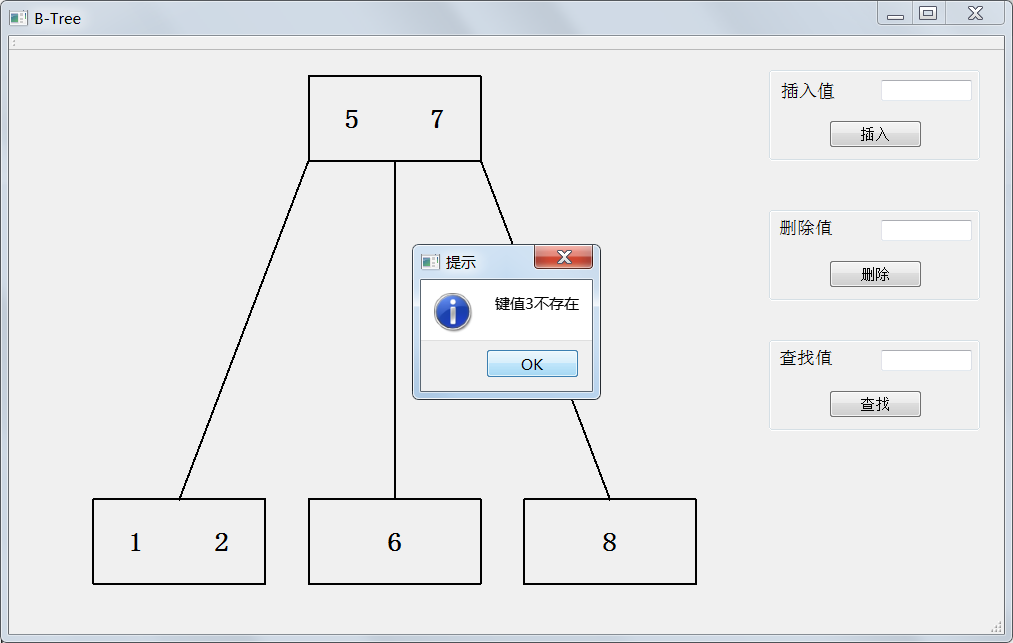


**1.6.2.2 容错能力**

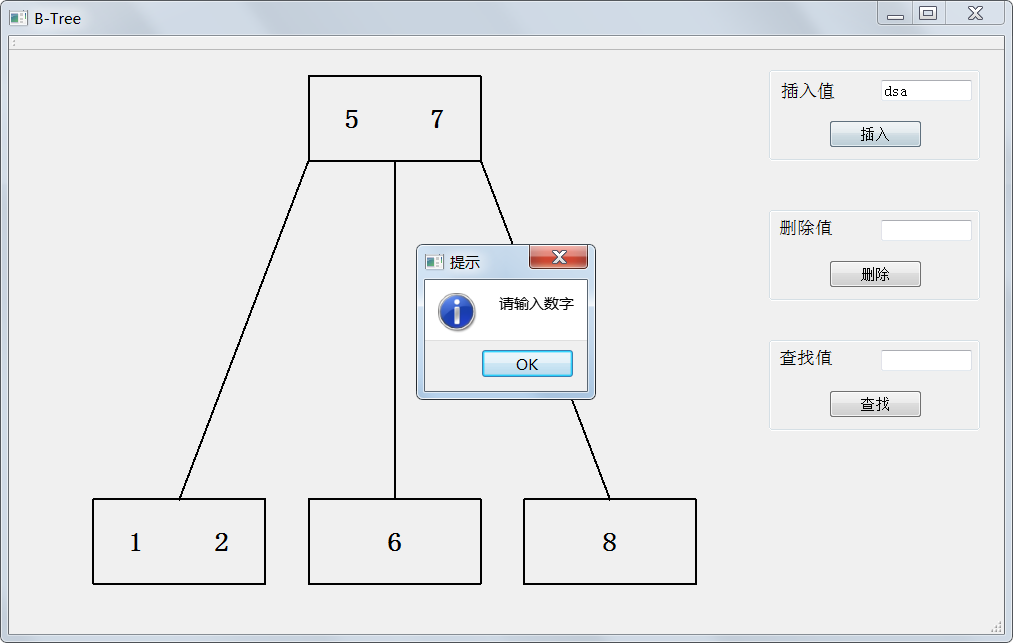
插入已存在的数值：



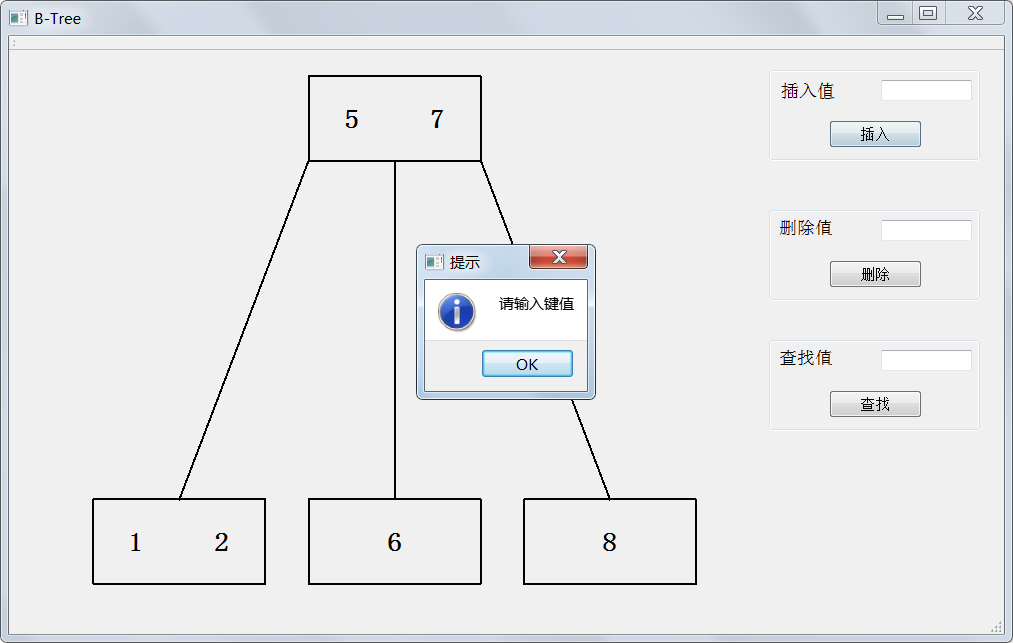
删除不存在的数值：



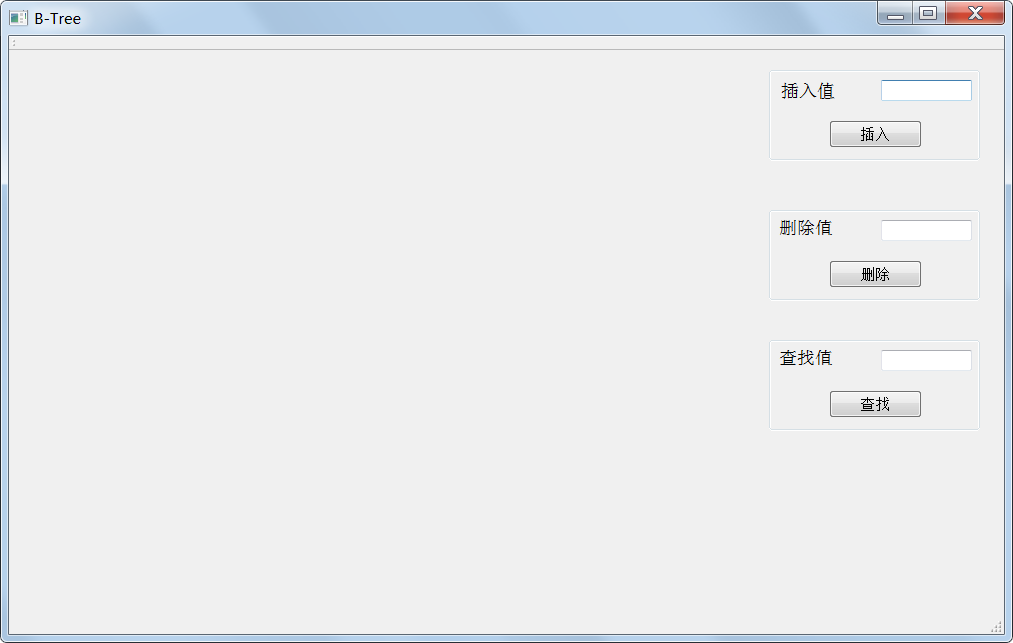
输入非数字字符：



不输入字符：



* 1. **操作说明**



在插入值的方框内输入要插入的值，然后点击插入按钮即可将值插入到B树中。

在删除值的方框内输入要删除的值，然后点击删除按钮即可将值删除。

在查找值的方框内输入要查找的值，然后点击查找按钮界面就会反馈查找结果。

**第二部分 综合应用设计说明**

1. 1. **题目**

题号2★★★：上海的地铁交通网络已经基本成型，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. 根据输入起点和终点站，显示地铁换乘指南。
3. 通过图形界面显示乘除路径。
   1. **软件功能**
      1. **图形化显示地铁网络**

（1）

class Station{

protected:

QString name;//站点名称

double longitude;//站点经度

double latitude;//站点纬度

};

将所有的站点信息储存在QVector<Station>stations中，将每个站点的经纬度按照等比例映射到视图坐标，在视图中对应的坐标上画一个圈，并在圈上标注站点信息，即可画出所有站点。

（2）

class Node{

public:

int stationID; //邻接点ID

int lineId; //两邻接点之间的线路ID

double distance; //两点距离

};

将地铁网络连线的信息储存在QVector<QVector<Node>> subwayNet中

class Line{

protected:

int id;//线路编号

QString name;//线路名称

QList<Station> stations;//途经站点

QColor color;//线路颜色

QString from;//起点站

QString to;//终点站

bool isLoopLine;//是否为环线

double transferTime;//发车间隔 min

double speed;//列车车速 km/h

};

将所有线路的信息储存在QVector<Line> lines中

对通过subwayNet中的信息进行处理，可以得到地铁网络所有连接的边（点对）QVector<QPair<int,int>>edges，然后在每对点间添加一条直线，直线的颜色为点之间线路的颜色。

* + 1. **动态添加地铁站点**

输入站点名称、站点经纬度信息，根据输入的信息，分配一个Station对象，将新增站点加入到QVector<Station>stations中，即可实现动态添加地铁站点。

* + 1. **动态添加地铁线路**

输入线路的名称，线路颜色，起点站，终点站，是否环线，途经站点，根据输入的信息分配一个Line对象，加入到QVector<Line> lines中，同时根据途经站点的信息更新QVector<QVector<Node>> subwayNet。

* + 1. **显示地铁乘车指南**

采取类似dijkstra算法的搜索方式，搜索从起点到目的点最短时间的乘车方案。

class SearchNode{

protected:

int stationId;//当前所到车站

int lineId;//到达车站所乘线路

double sumTime;//到达该车站所需时间

public:

SearchNode(int stationId,int lineId,double sumTime);

friend bool operator <(const SearchNode &one,const SearchNode &other);

friend bool operator >(const SearchNode &one,const SearchNode &other);

friend class SubwayNet;

};

QVector<QPair<int,int> > SubwayNet::getSubwayTransferGuide(QString stationName1,QString stationName2){

int stationId1 = stationsHash[stationName1];

int stationId2 = stationsHash[stationName2];

//time[i][j]记录从起点到station[i]且到达时乘坐列车id为j的最短时间

double \*\*time = new double\*[stations.size()];

for(int i = 0; i< stations.size();i++){

time[i] = new double[lines.size()];

for(int j = 0; j < lines.size(); j++){

time[i][j] = 99999999;

}

}

//trace[i]记录从起点到station[i/lines.size()]且到达时乘坐列车id为i%lines.size()时依次到达的站点id和到达站点时坐的列车id

QVector<QVector<QPair<int,int> > > trace;

for(int i = 0; i < stations.size()\*lines.size(); i++){

trace.push\_back(QVector<QPair<int,int> >());

}

priority\_queue<SearchNode,vector<SearchNode>,greater<SearchNode> > q;

q.push(SearchNode(stationId1,-1,0));

while(!q.empty()){

SearchNode searchNode = q.top();

q.pop();

int stationId = searchNode.stationId;

int lineId = searchNode.lineId;

double sumTime = searchNode.sumTime;

if(lineId >= 0 && sumTime > time[stationId][lineId]){

continue;

}

if(stationId == stationId2){

for(int i = 0; i< stations.size();i++){

delete []time[i];

}

delete[]time;

return trace[stationId\*lines.size()+lineId];

}

//扩展结点

for(Node node:subwayNet[stationId]){

int nextStationId = node.stationID;

int nextLineId = node.lineId;

double distance = node.distance;

Adouble nextSumTime = sumTime + distance / lines[nextLineId].speed \* 60 + (nextLineId == lineId?0:lines[nextLineId].transferTime);

if(nextSumTime < time[nextStationId][nextLineId]){

time[nextStationId][nextLineId] = nextSumTime;

if(lineId == -1){

trace[nextStationId\*lines.size()+nextLineId].push\_back(QPair<int,int>(nextStationId,nextLineId));

}

else{

int index = nextStationId\*lines.size()+nextLineId;

trace[index].clear();

trace[index].append(trace[stationId\*lines.size()+lineId]);

trace[index].push\_back(QPair<int,int>(nextStationId,nextLineId));

}

q.push(SearchNode(nextStationId,nextLineId,nextSumTime));

}

}

}

//没有从所输入起点到达终点的地铁线路

for(int i = 0; i< stations.size();i++){

delete []time[i];

}

delete[]time;

return QVector<QPair<int,int> >();

}

* 1. **设计思想**

class Station{

protected:

QString name;//站点名称

double longitude;//站点经度

double latitude;//站点纬度

public:

Station();

Station(QString n,double longi, double lati);

double calculateDistance(Station other);

QPointF getCoord();

friend class SubwayNet;

friend class MainWindow;

};

站点类的属性自然包含站点的名称，站点的位置（经纬度）信息。

class Line{

protected:

QString name;//线路名称

QList<Station> stations;//途经站点

QColor color;//线路颜色

QString from;//起点站

QString to;//终点站

bool isLoopLine;//是否为环线

double transferTime;//换乘等待时间 min

double speed;//列车车速 km/h

public:

//构造函数

Line();

Line(QString name, QColor lineColor,QString from,QString to,bool isLoopLine);

friend class SubwayNet;

friend class MainWindow;

};

* 线路类的属性自然含有线路的名称name。
* 线路颜色color是实现图形化界面时所需要的属性。
* 起点站、终点站是为了区别同一号线路产生分支的情况，这一号线路其实可以看成多条地铁线，只是有了一段比较长的公共路程，故有多少分支就用多个Line实例来储存这条线路。
* isLoopLine标志这条线路是否为环线，环线的特点即为起点站和终点站相连，即途经站点的首尾要互相连接。
* transferTime为换乘所需等待的时间，是在搜索最佳线路时需要考虑的因素。
* Speed是列车的车速，在搜索最佳线路时要计算到每一站的时间，需要用到该参数。

class Edge{

public :

int from;

int to;

QString lineName;

Edge(int fr,int t, QString l);

Edge();

friend bool operator ==(const Edge& one,const Edge& other);

};

Edge是为了方便线路的可视化而添加的类，记录着线路中的每一条连边，通过遍历这些连边，将其画在视图中，即可实现线路的可视化。

class Node{

public:

int stationID; //邻接点ID

int lineId; //两邻接点之间的线路ID

double distance; //两点距离

Node(int stationID,int lineId,double distance);

Node();

friend bool operator ==(const Node one, const Node other);

};

Node是图的邻接表表示结构中的结点，记录着邻接点和连边的信息（两邻接点之间的线路id，两点距离）

class SubwayNet{

protected:

QVector<QVector<Node>> subwayNet; //地铁线路网络图

QVector<Station> stations; //存储所有站点

QVector<Line> lines; //存储所有线路

QHash<QString, int> stationsHash; //站点名到存储位置的hash

QHash<QString, QVector<int>> linesHash; //线路名到存储位置的hash

QVector<Edge> edges; //所有边的集合

double maxLongitude = 0;//站点的最大经度

double minLongitude = 180;//站点的最小经度

double maxLatitude = 0;//站点的最大纬度

double minLatitude = 90;//站点的最小纬度

public:

QPointF getMinCoord();

QPointF getMaxCoord();

Status addStation(QString name,double longitude, double latitude);

QSet<QString> getLinesNameBetweenStation(int s1,int s2);

Status addLine(QString lineName,QColor color,QString from,QString to,bool isLoopLine);

Status addEdge(QString sta1,QString sta2,int lineId);

QVector<QPair<int,int> > getSubwayTransferGuide(QString stationName1,QString stationName2);

bool readFileData(QString fileName);

friend class MainWindow;

};

SubwayNet是管理整个地铁系统的类

* QVector<Station> stations记录着地铁网络的所有站点信息，可实现站点的可视化。
* QVector<Line> lines记录着地铁网络的所有线路的信息。
* QVector<QVector<Node>> subwayNet记录着地铁的图结构，在搜索最佳换乘路径时需要使用。
* QVector<Edge> edges;记录地铁网络所有边的集合，便于地铁网络连边的可视化。
* maxLongitude，maxLongitude，maxLatitude，minLatitude是为了记录经纬度坐标的边界，然后将所有的站点经纬度坐标转化为视图坐标时起作用
  1. **逻辑结构与物理结构**

class Station{

protected:

QString name;//站点名称

double longitude;//站点经度

double latitude;//站点纬度

public:

Station();

Station(QString n,double longi, double lati);

double calculateDistance(Station other);

QPointF getCoord();

friend class SubwayNet;

friend class MainWindow;

};

class Line{

protected:

int id;//线路编号

QString name;//线路名称

QList<Station> stations;//途经站点

QColor color;//线路颜色

QString from;//起点站

QString to;//终点站

bool isLoopLine;//是否为环线

double transferTime;//发车间隔 min

double speed;//列车车速 km/h

public:

//构造函数

Line();

Line(int id,QString name, QColor lineColor,QString from,QString to,bool isLoopLine);

friend class SubwayNet;

friend class MainWindow;

};

class SubwayNet{

protected:

QVector<QVector<Node>> subwayNet; //地铁线路网络图

QVector<Station> stations; //存储所有站点

QVector<Line> lines; //存储所有线路

QHash<QString, int> stationsHash; //站点名到存储位置的hash

QHash<QString, QVector<int>> linesHash; //线路名到存储位置的hash

QVector<Edge> edges; //所有边的集合

double maxLongitude = 0;

double minLongitude = 180;

double maxLatitude = 0;

double minLatitude = 90;

public:

QPointF getMinCoord();

QPointF getMaxCoord();

Status addStation(QString name,double longitude, double latitude);

QSet<QString> getLinesNameBetweenStation(int s1,int s2);

Status addLine(QString lineName,QColor color,QString from,QString to,bool isLoopLine);

Status addEdge(QString sta1,QString sta2,int lineId);

QVector<QPair<int,int> > getSubwayTransferGuide(QString stationName1,QString stationName2);

bool readFileData(QString fileName);

// bool isTransferStation(QString station);//判断是否为换乘站

// bool isTransferStation(int station);

friend class MainWindow;

};

* 1. **开发平台**

开发语言 ：C++（C++11标准以上）

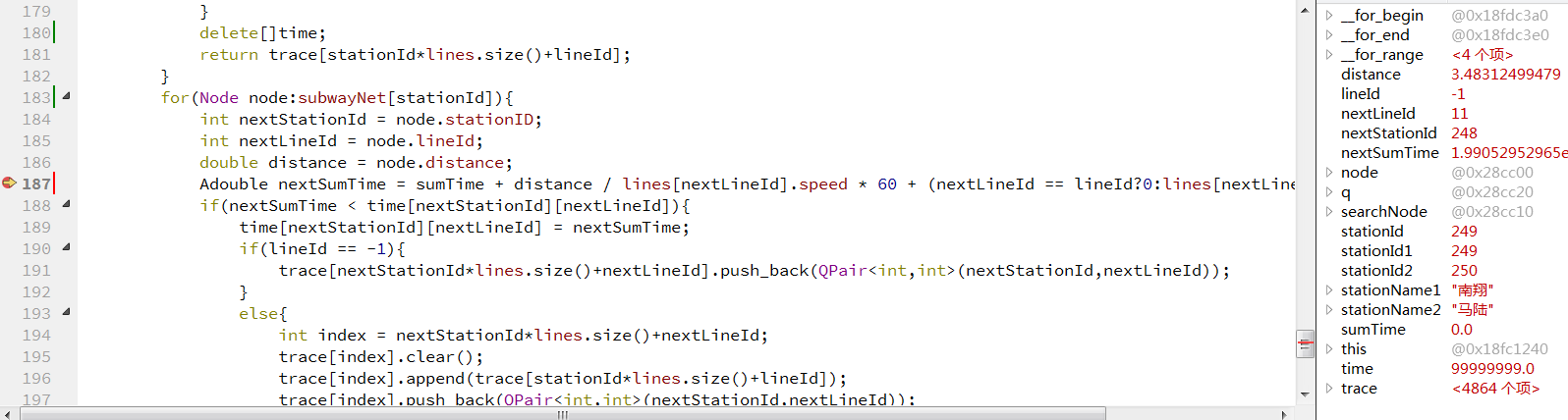
开发框架 ：QT

集成开发环境：Qt Version 5.9.0

编译器 ：MinGW 32bit

* 1. **系统的运行结果分析说明**

**2.6.1 调试及开发过程**

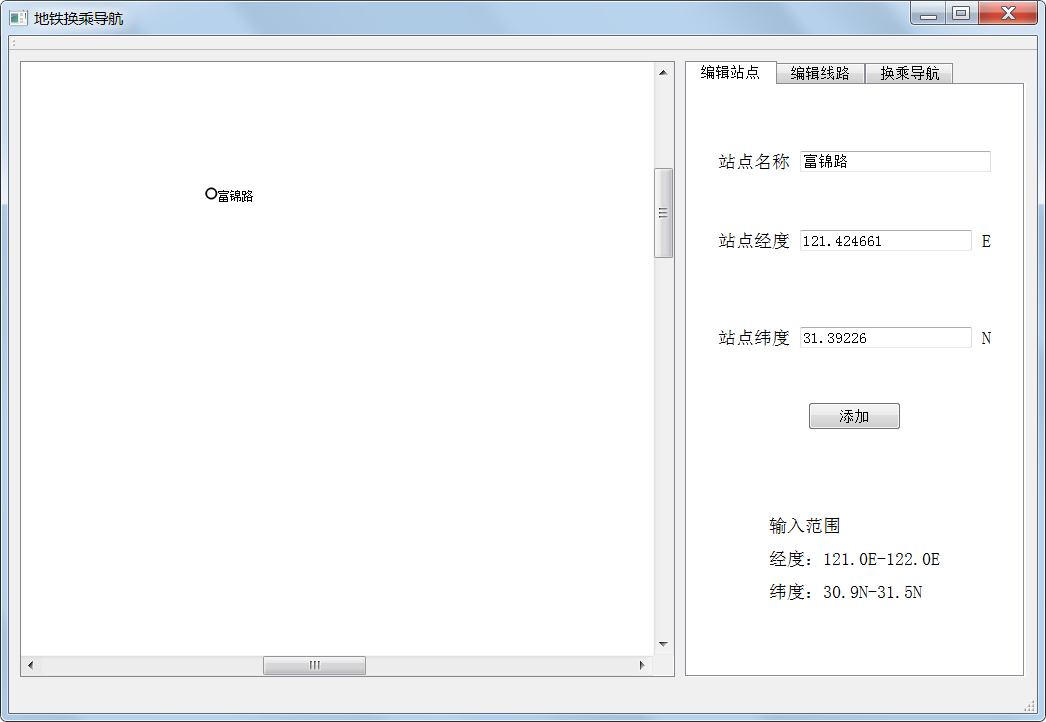


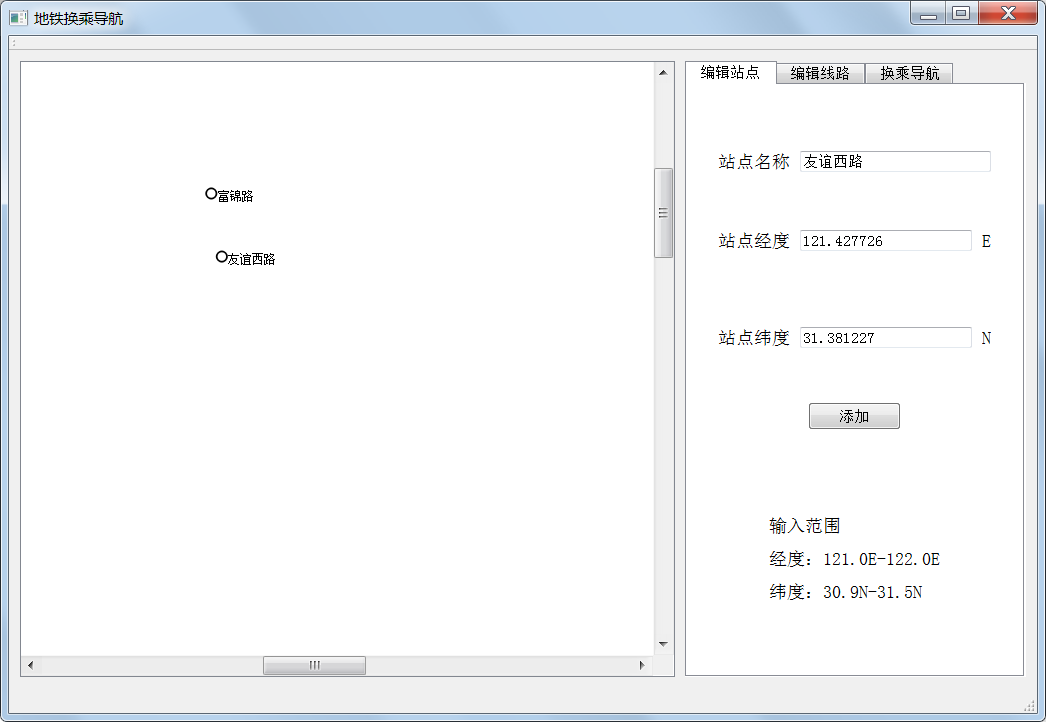
QT提供Debug工具，快捷键F5，代码左端点击设置断点，在右侧可以查看各个变量的值

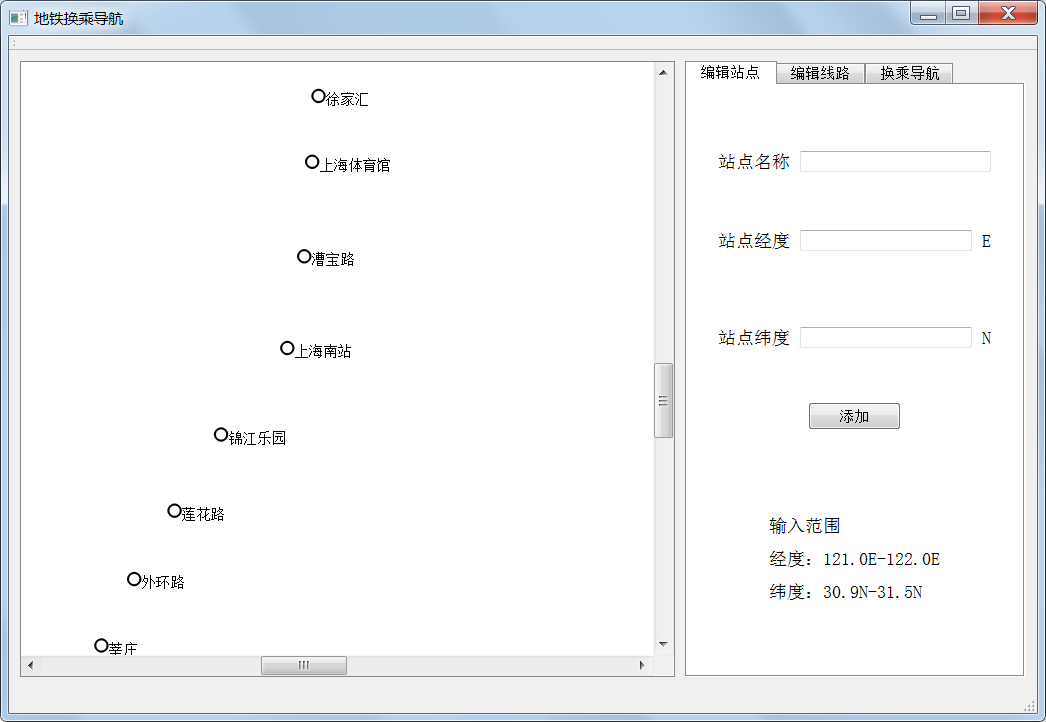
**2.6.2 成果分析**

**2.6.2.1 正确性**

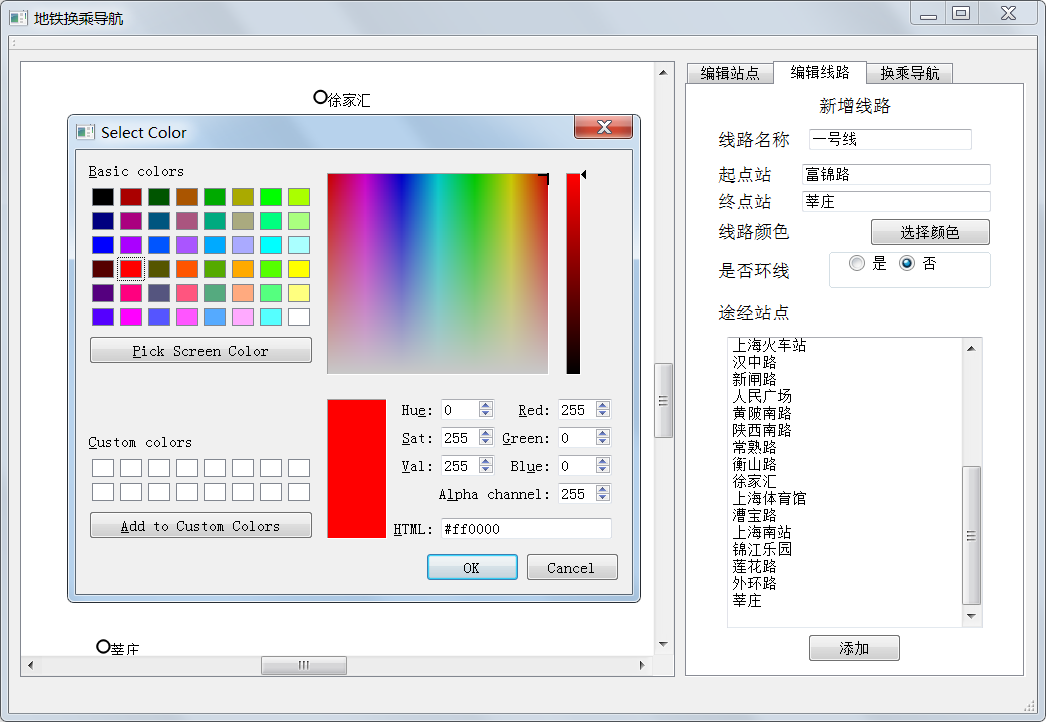
**添加站点**

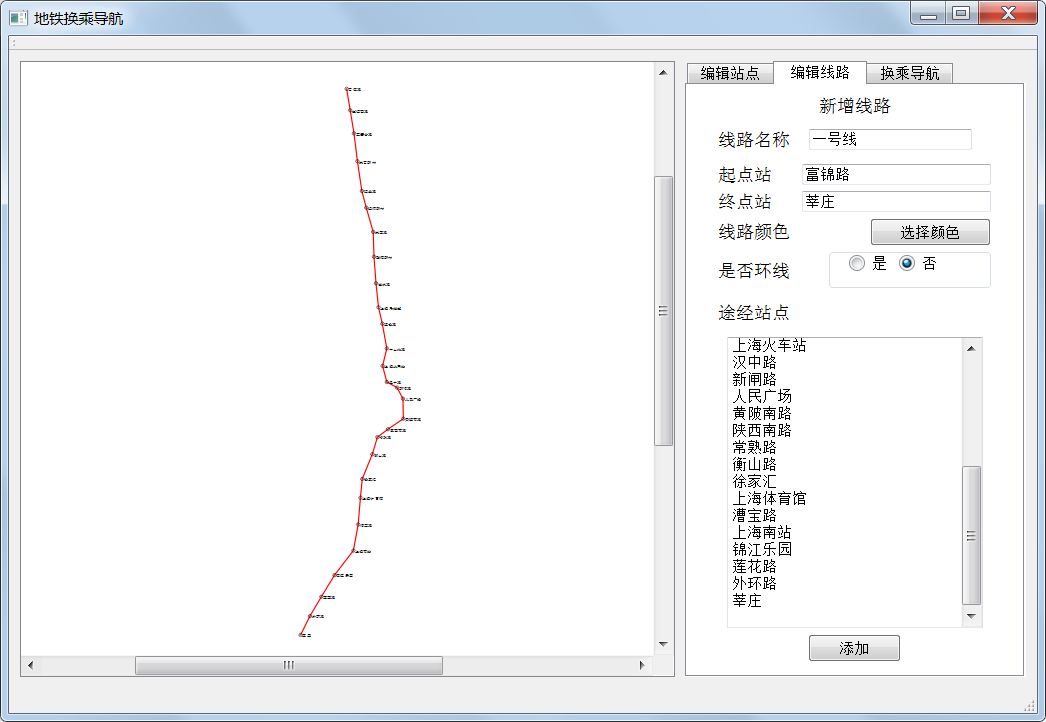




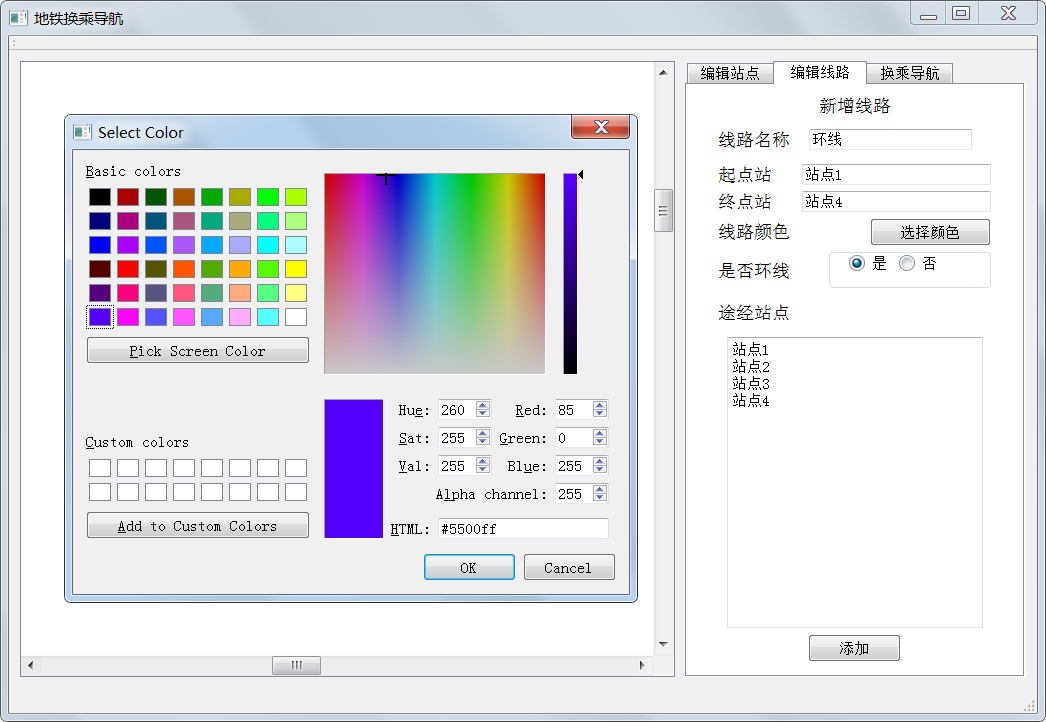
****

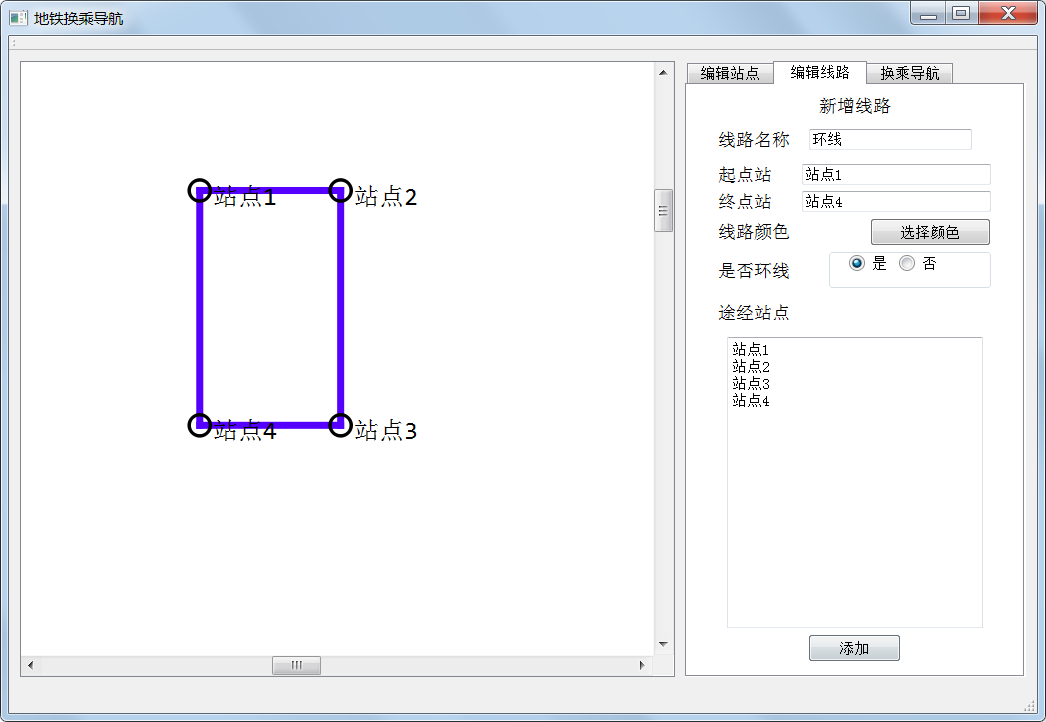
**添加线路**



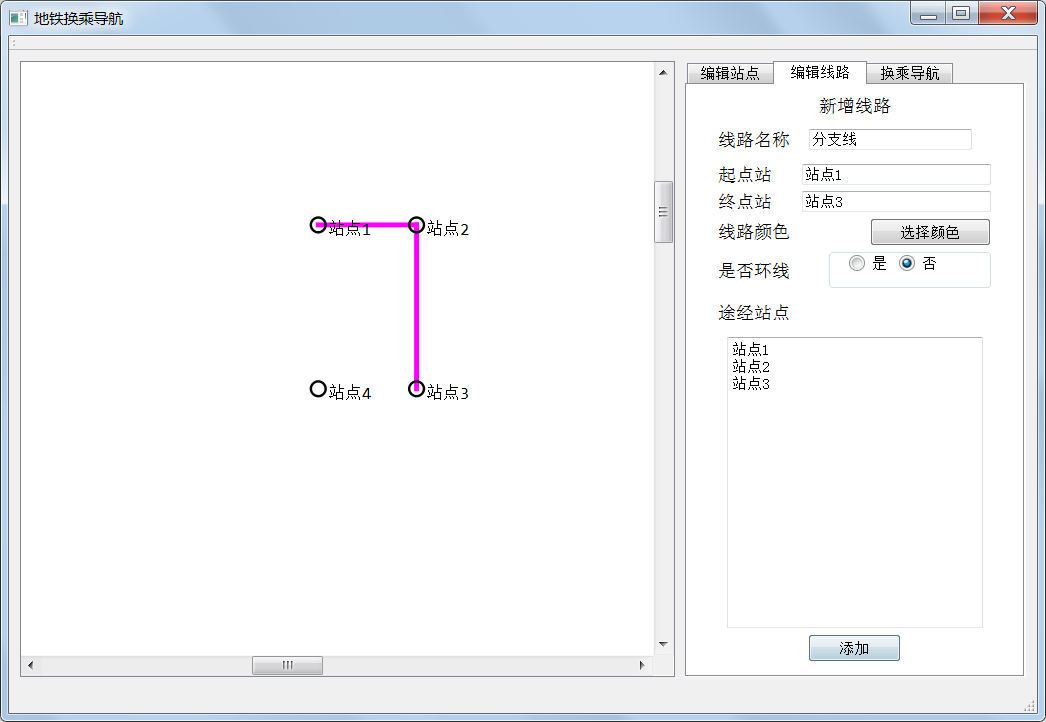


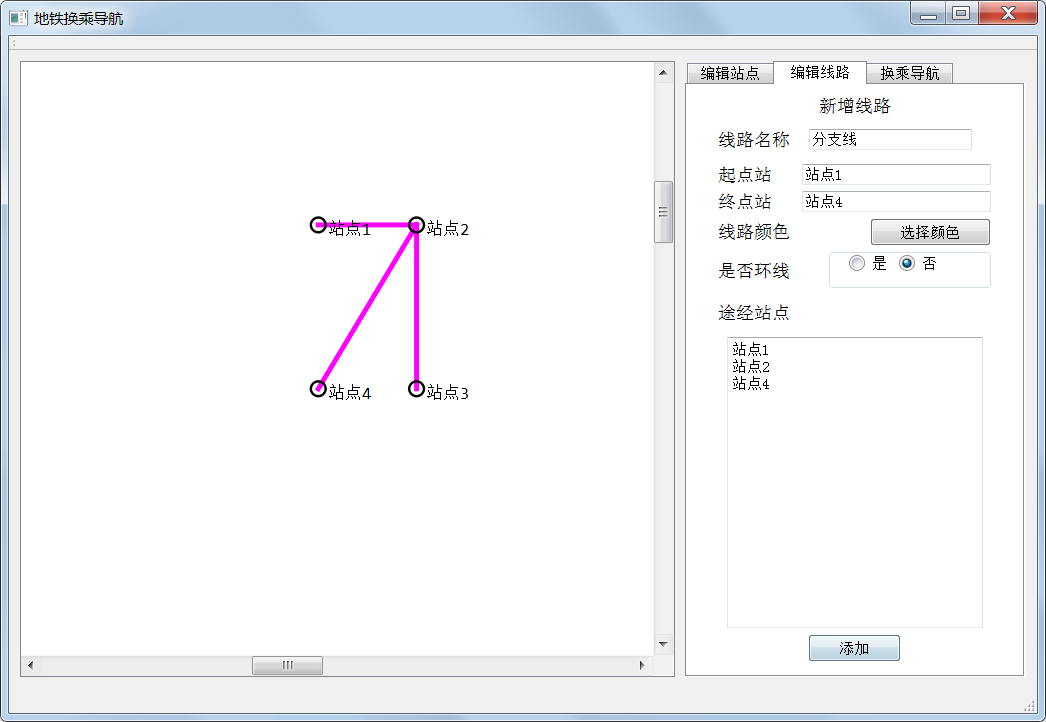
**添加环线**



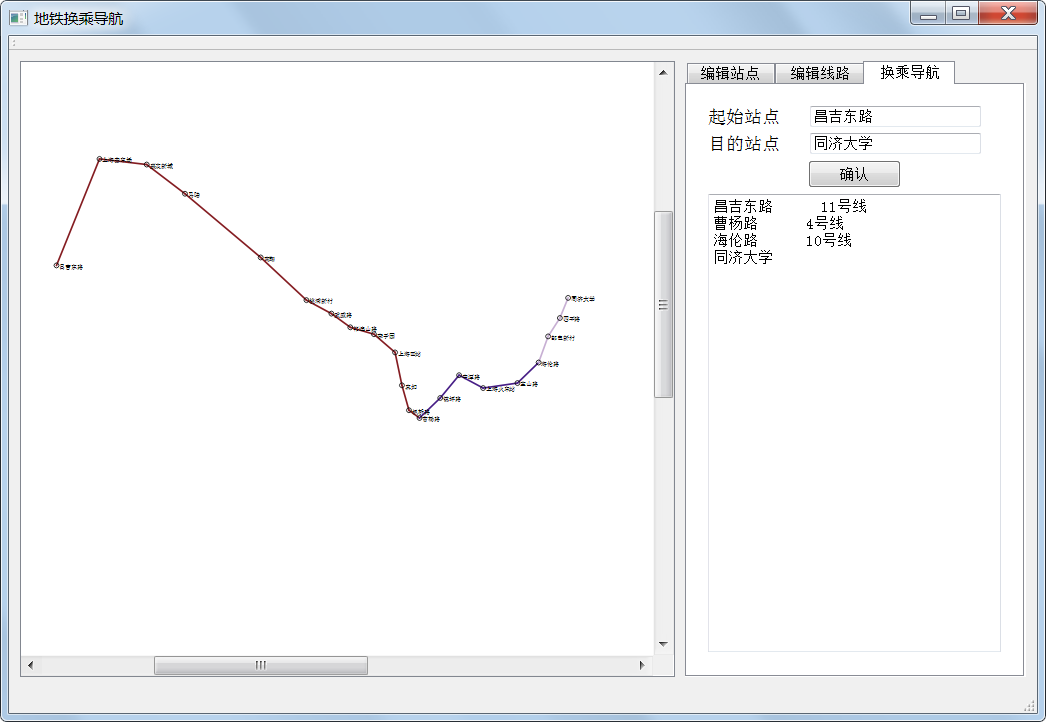


**添加有分支的线路**



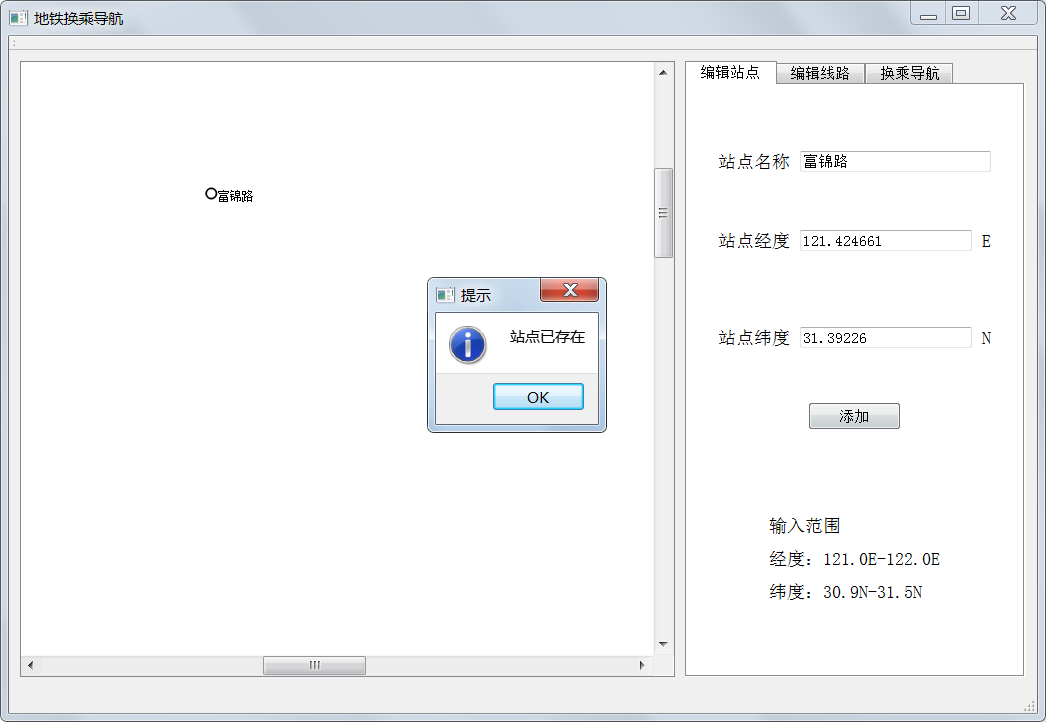


**乘车指南**

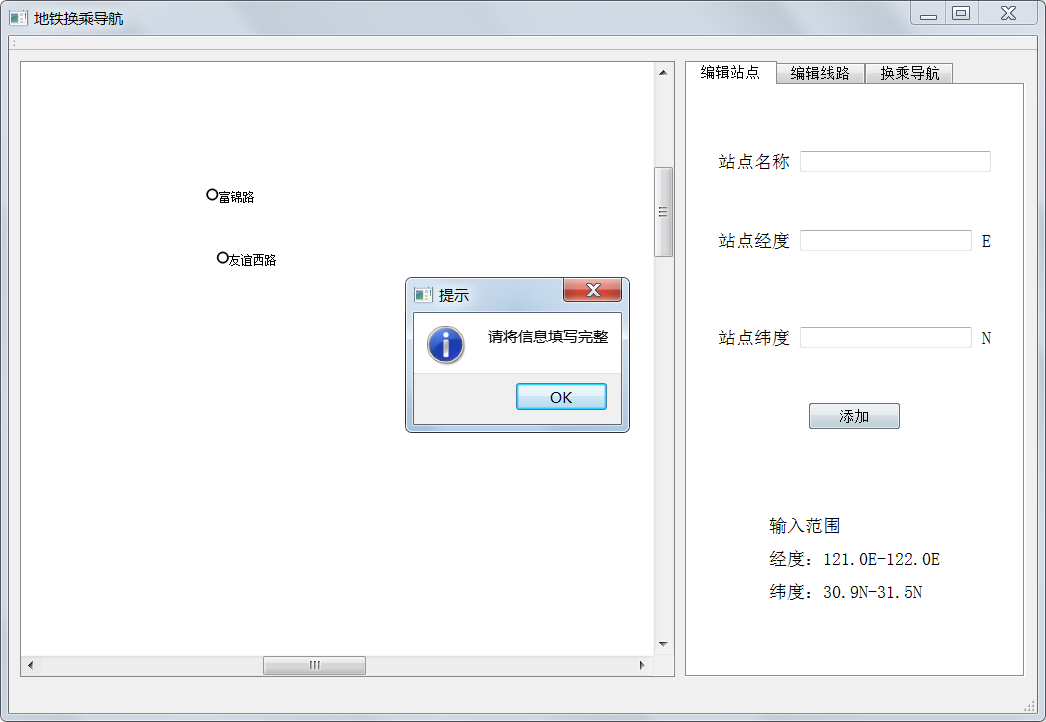


**2.6.2.2 容错能力**

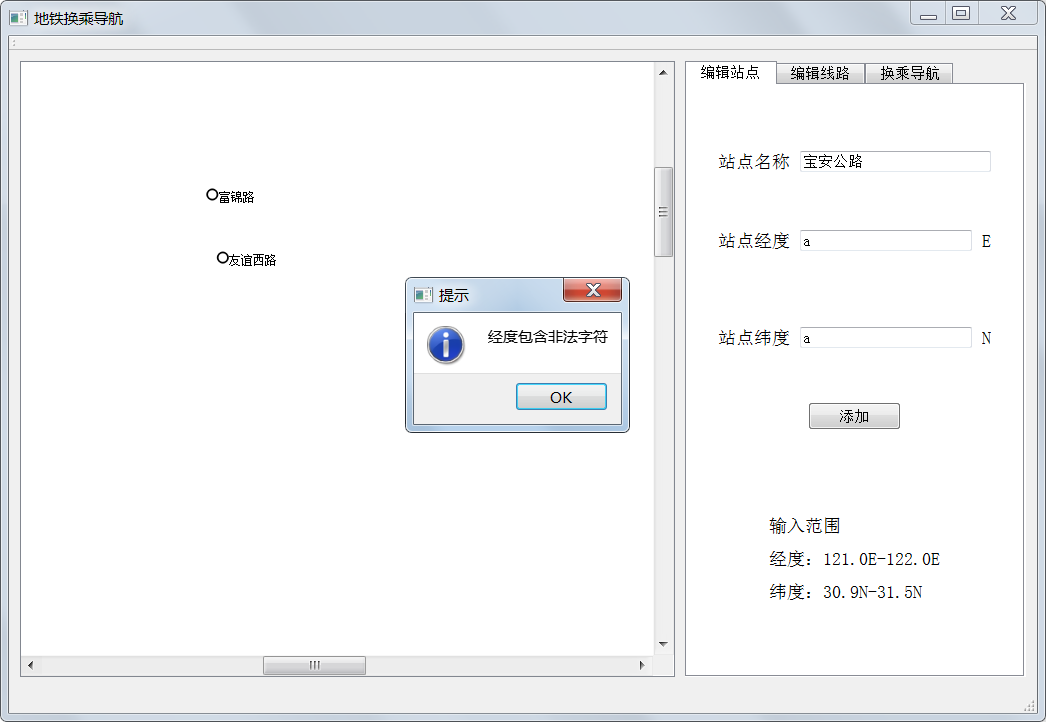
**重复添加站点**



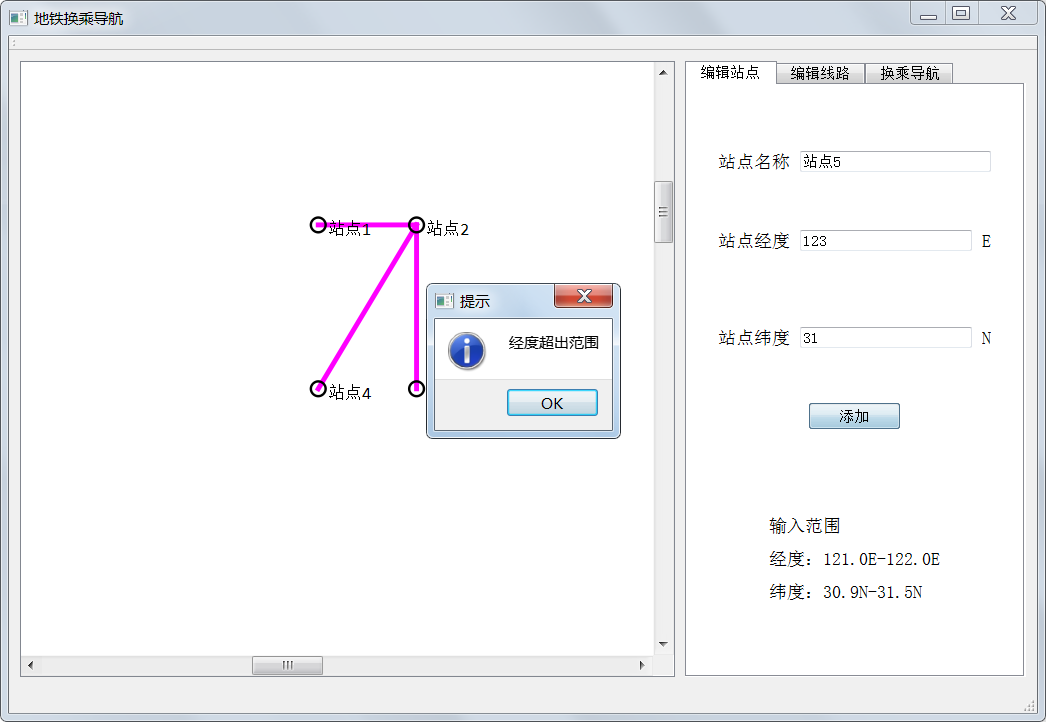
**不输入信息直接添加**



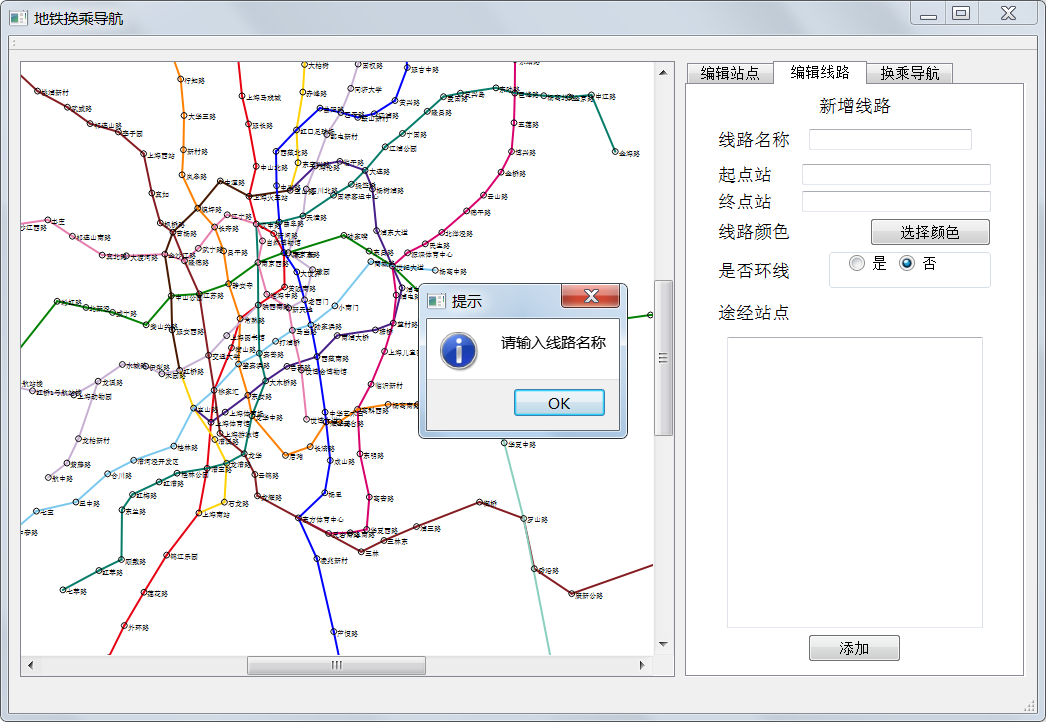
**经纬度中含有非法字符**



**经纬度超过范围**



**添加线路中没有输入线路名称**



**添加线路中没有选择是否环线**



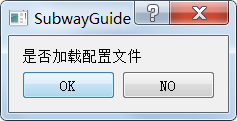
**添加线路中包含不存在站点**



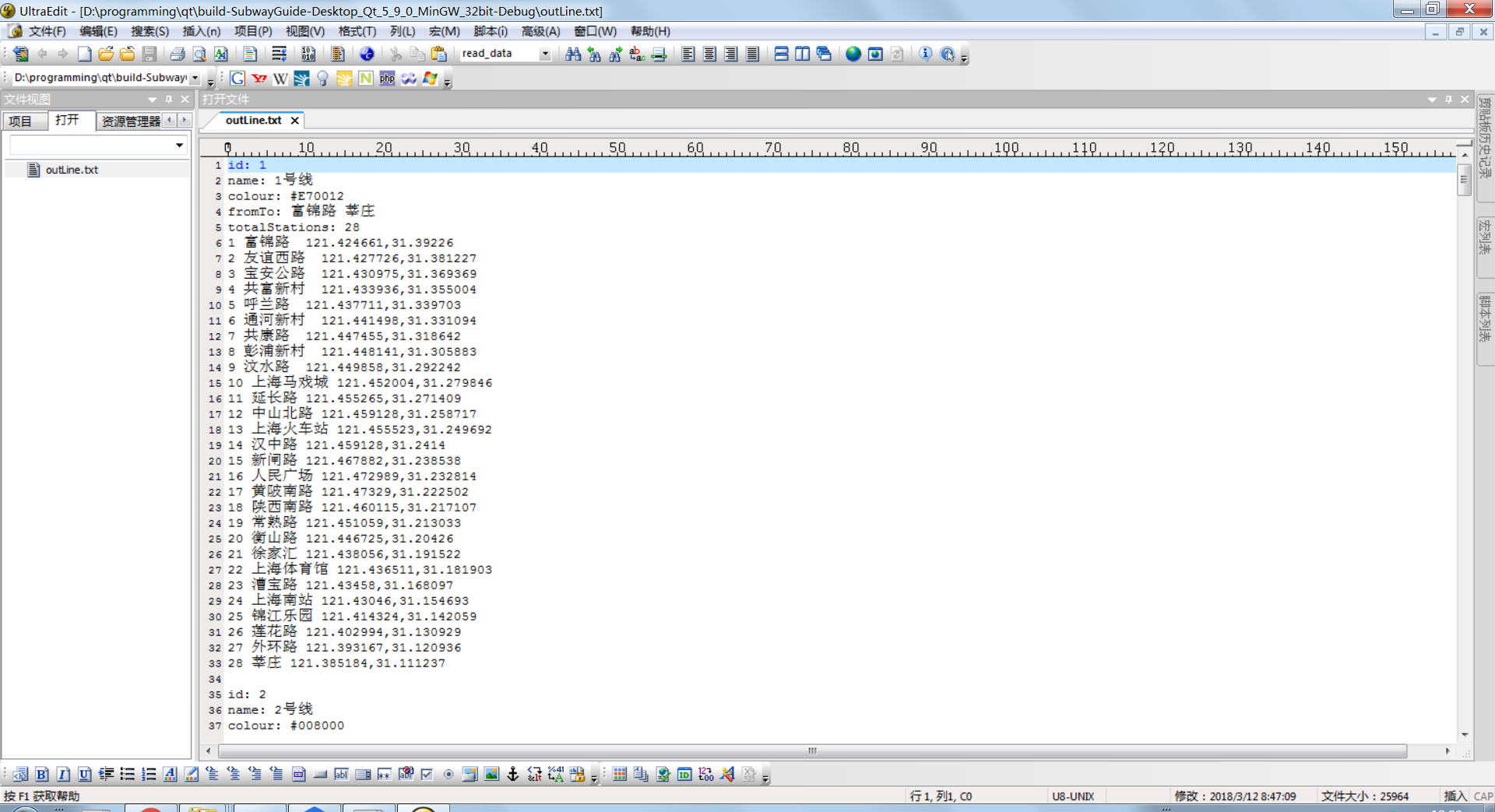
**输入不存在站点**



* 1. **操作说明**
     1. **初始化**



在启动程序时，会询问你是否加载配置文件，该配置文件为与exe文件同目录的outline.txt

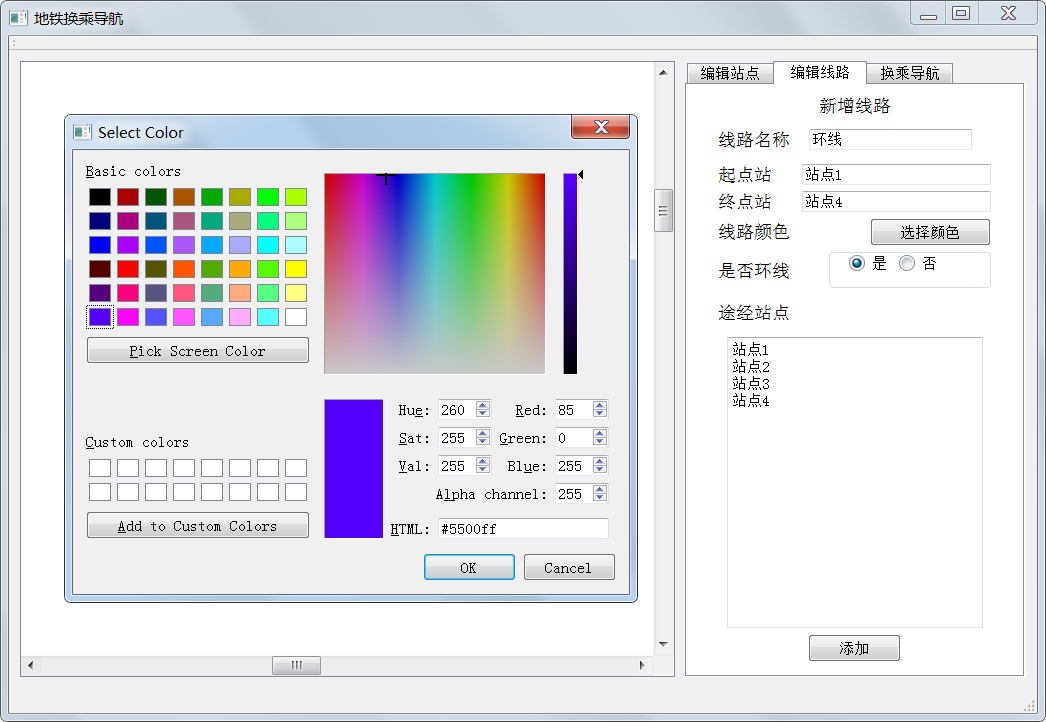


**2.7.2 添加站点**



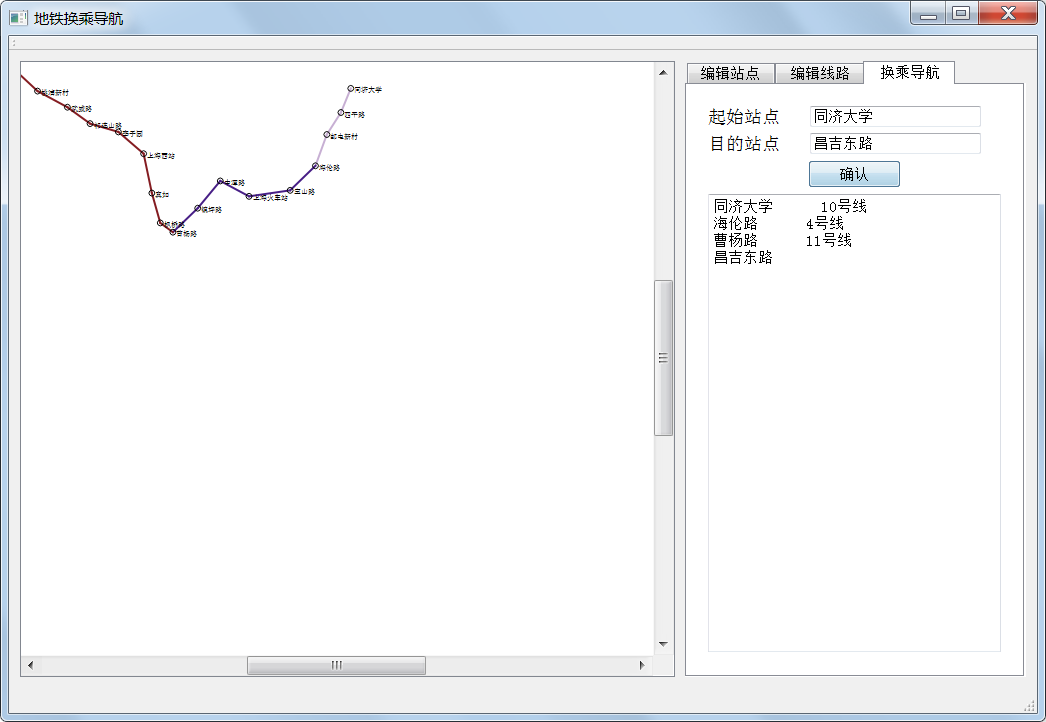
输入站点名称、站点经度，站点纬度，点击添加即可。

**2.7.3 添加线路**



输入线路名称、起点站、终点站、选择线路颜色，是否环线、输入途经站点，点击添加即可完成

**2.7.4 查询乘车方案**



输入起点站、终点站点击确认即可获取乘车方案。

**提示：**如果要重新查看地铁网络图，将换乘导航切换到编辑站点或编辑线路即可。

**第三部分 实践总结**

1. 1. **所做的工作**

学习和运用了Qt5的编程架构和相关知识；

学习了Github平台项目托管的运用；

独立完成课程设计中的算法实现题目的所有设计、编程实现、测试等工作；

独立完成课程设计中的综合应用题目的所有设计、编程实现、测试等工作；

完成课程设计报告一份

* 1. **总结和收获**

**能力提升：**

完成数据结构课程设计后，感觉自身的自学能力有了很大提升。在课程设计以前，基本上所有的编程知识都是由老师言传身受、一点一滴的对我们进行指导，我们自学的东西较少，自学能力也锻炼的少。但是这次的课程设计的所有工作，都是由我们自己去选择相应的技术学习实现，仅仅掌握基本编程语言的我们通过自学来完成这次课程设计。在这个过程中，大学生尤其是计算机专业的大学生所需要的自学能力得到了必须的锻炼，对以后所有的技术和知识的自学，这将是一个有意义的开端，现在我们已经掌握了如何快速学习一门技术的能力，对以后学习和工作中的所有挑战做好了准备。

**收获：**

虽然课程设计只有看似算法设计和综合应用实现两个题目，但是从布置课程设计题目到完成的整个过程中，不仅仅是学习到了以前未曾接触过的标准用户交互程序，从简陋的控制台程序提升到界面友好、美观、人性化的应用程序，打开了编程的一扇大门，同时也从心态、自学能力、解决未知问题能力等内在的综合实力有了全新的提升。具体来说，学习了Qt的编程框架，包括元对象机制、基本控件的掌握、绘图框架、模型编程、文件操作等相关知识；学习到了Github开源项目托管平台的使用，将其运用到了实际的项目；学习到了openstreetmap等开源地图数据的获取和提炼等知识，自身技术能力又上了一个台阶。在软实力上，打开了用户界面编程的大门，激发自身对编程的热情和后面学习的动力；通过自学实现了课程设计，自身的学习能力得到了有效锻炼；开发的过程中，无论是设计还是编程，都会遇到各种各样的问题，通过独立解决这些疑难杂症，在独立解决问题的能力方面也得到了很大提升；最重要的是，树立不惧所有未知知识、解决疑难问题、接受各种挑战的信心。

**个人体会：**

在学习和实现的过程中体会和领悟还是比较的多，这里挑出感触最深的几点吧。第一，做事情一定要找最重要的事情先做，这样做事情才有效率，不拖拉。由于个人比较追求完美，对程序的用户界面和一些细节很是用心，但是这样就好浪费很多时间，效率不高，应该先做好最重要的事情，例如最不可缺少的功能实现，主体实现然后细节修改，而不是本末倒置。第二，学到的东西不一定都用的到，所以开发还是有一些功利心较好，用到什么然后去学习相应的知识并用到实处，这样才能学以致用，恰到好处。在一开始的学习中，每天看书看教程，发现过去了几天还是什么都不会，然后边学边做，才领悟到学习方法和要点。

**第四部分 参考文献**

1. 霍亚飞.Qt Creator快速入门第二版[M].北京：北京航空航天大学出版社,2014-1
2. 霍亚飞，程梁. QT5 编程入门[M]. 北京：北京航空航天大学出版社,2015-1
3. 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