華中科技大學

课 程 实 验 报 告

课程名称：C++程序设计

实验名称：面向对象的公交地图导航

院　　系 ：计算机科学与技术

专业班级 ：　计科 2011 班

学　　号 ：　U202015084

姓　　名 ：　张文浩

指导教师 ：　金良海

2021 年 12 月 17 日

一、需求分析

1. 题目要求

假定所有公交车辆从起点到终点都是双向非环路的，且双向线路的所有停靠站点都对应相同。设有 M 路公交路线，第 j 路公交路线有 N_j 个站点。所有公交线路累计共有 S 个站点，第 k 个站点的坐标为（X_k，Y_k），所有坐标均以米为单位标注。邻近站点之间的距离指的是站点坐标之间的欧几里得距离。现有一人处于起点坐标（X_b，Y_b），此人需要步行到最近站点乘车，下车后要步行到达的终点坐标为（X_e，Y_e），而他特别不愿意走路，能坐公交就尽量坐公交。假定公交转乘时的步行距离为 0，试编程求他从起点（X_b，Y_b）到终点（X_e，Y_e）的乘坐线路。建立模型时需要考虑如下几种情形：（1）最少转乘；（2）最短距离。

所有公交线路的站点坐标存放于“stops.txt”文件，其中第 1 行为站点总个数，第 2 行为第 1 个站点的坐标，第 3 行为第 2 个站点的坐标，以此类推。可用图形化的界面显示站点及公交线路。“stops.txt”文件的内容如下。

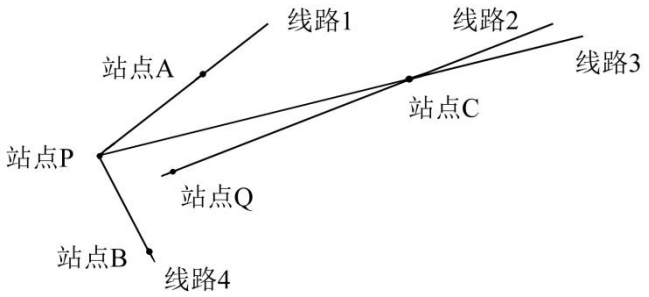
```
39
235     27
358     29
480     34
155     36
222     64
282     62
413     60
457     63
483     60
560     69
131     87
349     61
314     97
420    107
487    125
620    107
666     79
186    107
270    120
350    141
383    148
370    164
```

442 179
496 171
555 167
651 155
775 184
678 272
208 156
296 161
356 190
493 202
490 229
504 262
457 269
249 196
155 190
103 171
112 241

所有公交线路信息存放于“lines.txt”文件，其中第 1 行为公交线路总数，第 2 行为每条公交线路的站点总数，第 3 行为线路 1 经过的站点编号（对应站点坐标参见“stops.txt”），第 4 行为线路 2 经过的站点编号，以此类推。“lines.txt”文件的内容如下。

6
13 11 8 9 7 7
1 6 13 20 22 21 14 8 3 9 15 24 32
4 5 6 12 7 8 9 10 16 26 28
11 18 19 20 21 23 33 34
38 37 36 31 23 24 25 26 27
2 12 13 19 29 37 39
30 31 35 33 25 16 17

采用 Dijkstra 最短路径算法是不合适的，因为它仅考虑了距离而未考虑公交线路行驶约束。如下图所示，对于某站点 P 周围的站点 Q，其欧几里得距离虽近，但可能需要很远的转乘才能到达。例如，从站点 P 出发，要从线路 3 经站点 C 转线路 2 才能到最近的站点 Q，因为站点 P 和站点 Q 之间无直达的公交线路。



通过类型抽象形成站点、公交线路、转乘站点、转乘线路、转乘矩阵、公交系统等类，输入上述文件初始化站点和线路对象，然后通过图形化的界面显示站点和线路地图。用户用鼠标左键在地图上设定起点、鼠标右键确定终点，按照设定的最少转乘或者最短距离选项规划出从起点步行到最近站点上车、到离终点最近站点下车步行到终点的线路，在地图上用不同颜色显示规划线路若干秒，然后消除并恢复原始地图线路的颜色。

假设某个机构或单位的信息存储在“organization.txt”文件，第 1 列存放单位名称，第 2 列单位坐标。“organization.txt”文件中的格式如下，可自己添加更多单位或坐标。

华中科技大学　　990, 370
华乐山庄　　500, 340
光谷中心花园　　631, 367
光谷街北路　　766, 472

用户可输入“华科大”或“华中科大”查询其所在位置，通过最大公共子串算法进行模糊匹配，找到“华中科技大学”及其坐标。在确定始发单位坐标和终点单位坐标后，按照设定的最少转乘或者最短距离选项规划线路，并在地图上用不同颜色显示规划线路若干秒，然后消除并恢复原始地图线路的颜色。

提示：可以构造公交线路转乘矩阵 A（即 A^1）。假定有 5 条公交线路，若线路 i 和线路 j（i≠j）有 r 个转乘站点（即 r 种走法），则矩阵 A^1 的元素 $a^1[i,j]=r$；如 i=j 则规定无须转乘，即有 $a^1[i,i]=0$。则对于上述前 5 条公交线路，从公交线路 i 一次转乘到线路 j，可得如下转乘矩阵 A^1。

0　3　3　1　0
3　0　3　2　1
3　3　0　0　1
1　2　0　0　1
0　1　1　1　0

由上述转乘矩阵 A^1 可知，无法从线路 1 转乘到线路 5，因为 $a^1[1,5]=0$。可以尝试从线路 1 转乘其他线路，再从其他线路转乘线路 5，即从线路 1 经过两次转乘到线路 5。这只需要进行矩阵乘法运算 $A*A=A^2$，从线路 1 经过两次转乘到线路 5，可从 A^2 中得到共有 $a^2[1,5]$种走法。

$$a^2[1,5]=\sum_{k=1}^{5}a^1[1,k]*a^1[k,5]$$

由上述转乘公式，可计算得到 A^2，最后设置 $a^2[i,i]=0$，修正 A^2 使同一线路不用转乘。修正后的 A^2 如下。

0	11	9	6	7
11	0	10	4	5
9	10	0	10	3
6	4	10	0	2
7	5	3	2	0

由上述矩阵可知 $a^2[1,5]=7$，即从线路 1 经过两次转乘到线路 5 共有 7 种走法：（1）从线路 1 到线路 2 共有 3 个转乘点，再从线路 2 经唯一转乘点至线路 5，一共有 3 种转乘方法；（2）从线路 1 到线路 3 共有 3 个转乘点，再从线路 3 经唯一转乘点至线路 5，一共有 3 种转乘方法；（3）从线路 1 经唯一转乘点至线路 4，再从线路 4 经唯一转乘点至线路 5，一共有 1 种转乘方法。

依此类推，可以得到 A*A*A（即 A^3，反映从线路 i 经过 3 次转乘到线路 j 共有几种转乘走法）。对于本题来说，由于总共只有 M=7 条公交线路，故无重复公交的转乘最多只能转乘 M-1 次，所以只需计算到 $A^{M-1}=A^6$ 为止。任意两条线路之间 1 次、2 次……6 次转乘的走法共有 A^+种，$A^+= A^1+ A^2+ A^3+ A^4+ A^5+ A^6$。

上述 A^+运算是一种修正的矩阵“闭包”运算，可以抽象成矩阵类的一个运算。上述 A^+运算只计算了有几种转乘走法，当然，还可以同步 A^n 建立另外一个矩阵，对应元素为某种链表指针类型，用于记载对应转乘线路和转乘站点。

得到上述转乘“闭包”矩阵 A^+以后，只需要搜索离起点最近的站点（可能不止一个站点），找到最近站点所在的线路 i（可能不止一条），并搜索离终点最近的站点（可能不止一个站点），找到最近站点所在的线路 j（可能不止一条），然后在 A^+中找出 a[i, j]所描述的所有转乘线路和转乘站点即可。利用站点坐标可以得到两两站点之间的距离，并利用 A^+分别得到如下两种情形的最优转乘方案：（1）最少转乘；（2）最短距离。

2. 需求分析

本次实验需要用 C++实现公交地图导航功能，需要读取已有的站点、线路、地点文件，选取起点和终点，利用合适的算法，寻找起点和终点之间需要最少转乘的线路，并在地图上展示。

二、系统设计

1. 概要设计

本次实验需要用 C++实现公交地图导航功能并利用 Qt 设计界面，总体结构包括数据结构模块和 Qt 界面模块。

数据结构模块：

本次实验设计了公交站点类（包括站点编号、站点坐标）、公交线路类（包括线路编号、

所有站点编号、站点数量）、转乘站点类（包括现在线路、要转乘线路、站点编号）、转乘路径类（包括转乘路径上的所有转乘站点以及转乘站点的个数）、转乘次数和线路类（包括转成路径方案数组以及方案数量）、转乘矩阵类（包括矩阵指针和矩阵的行数列数）以及总括类（包括公交站点及数量、公交线路及数量、转乘矩阵等）以及类的一系列成员函数。

各个类的成员函数如下：

POINT：获取编号、坐标

LINE：初始化、赋值运算符重载、判断线路中是否含有某站点、判断两线路是否相交、取线路编号、取线路中站点的数量、取线路中某个站次的站点编号、析构

CROSS：初始化、判断相等运算符重载、获取线路编号、站点坐标

OPTION：初始化、得到转乘次数、判断相等运算符重载、去重复转乘、取某一转乘、添加线路（+、+=运算符重载）、赋值运算符重载、析构

NODE：初始化、去掉相同转乘、添加路径、连接路径、赋值、获取转乘路径数目、打印元素、析构。

MAT：初始化、查找是否有不可到达站点、获取最少转乘路径数、获取 NODE、获取转乘路径数目、加法重载、乘法重载、添加路径、打印矩阵、析构

TOTAL：初始化、获取最少转乘方案数、析构

其中核心算法为 MAT 中的 minitrans 函数以及 TOTAL 类的初始化函数，将在下一部分详细介绍。

Qt 界面模块：

设计的 Qt 模块包括：

QtTipsDlgView，鼠标移动到站点的位置时显示站点信息。

QtWidgetsFL，站点文件和线路文件加载窗口

QtWidgetsOrg，地点文件加载窗口

QtWidgetsBE，起点和终点选择窗口

map，显示地图、站点、线路以及进行读取文件选择地点获取最少转乘路径等操作

涉及到的槽函数有：

map：loadmap(),mintrans(),closewnd(),loadorg(),btoe()

QtWidgetsFL：inputStop(),close(),inputLine(),checkFile()

QtWidgetsOrg：close(),inputOrg(),checkFile()

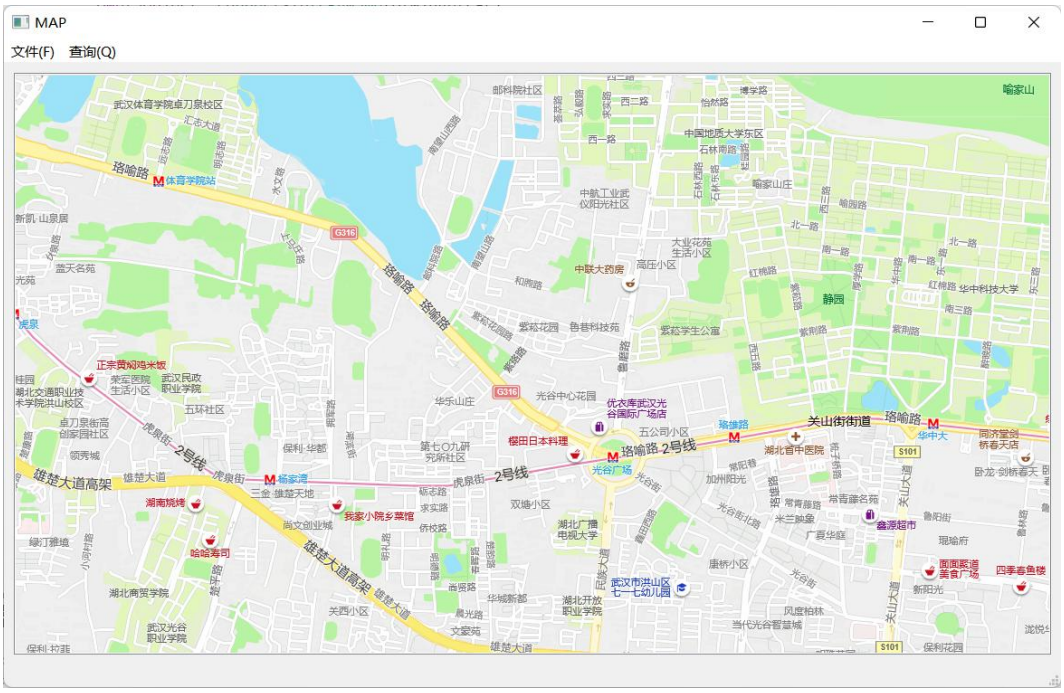
QtWidgetsBE：close(),Done()

通过槽函数实现界面与数据结构之间的连接，操作 Qt 界面按下每一个功能时会发出响应信号，信号发射之后会调用相应的槽函数，然后槽函数再调用数据结构中的类成员函数，实现两个模块的连接。

详细的函数调用关系图在“详细设计”中会展现。

图 2-0　函数调用关系

界面设计：



(“文件”中有选项读入地图、读入地点、输入起点和终点，“查询”中有选项最少转乘)

图 2-1-1 map.ui

图 2-1-2　文件

图 2-1-3　查询

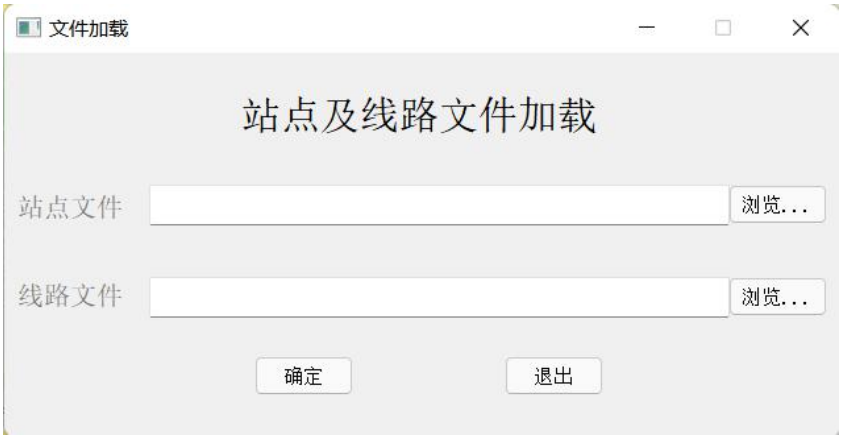


图 2-2　QtWidgetsFL.ui

图 2-3　QtWidgetsOrg.ui

图 2-4　QtWidgetsBE.ui

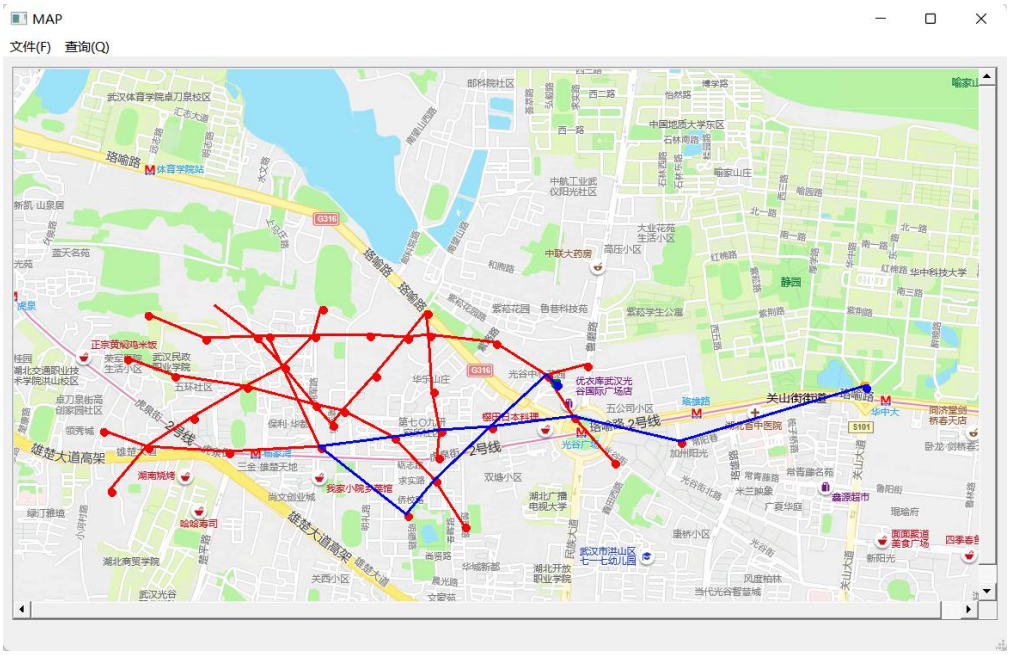


图 2-5　效果图

总体流程图如下：

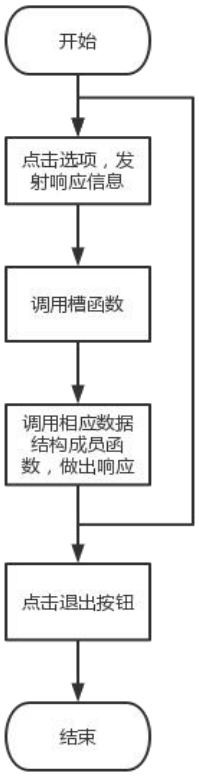


图 2-6　总体流程图

2. 详细设计

数据结构模块：

①站点类：

```
class POINT {              //公交站点
    int num;              //编号
    int x, y;             //坐标
public:
    POINT(int n=0, int x=0, int y=0);
    int& getx();
    int& gety();
    int& getnum();
};
```

②线路类：

```
class LINE{               //公交线路
    int num; //编号
    int* stop; //所有站点编号
    int count; //站点数量
public:
    LINE(int num = 0, int count = 0, int* stop = 0);
    LINE(const LINE& r);
    LINE(LINE&& r) noexcept;
    LINE& operator=(const LINE& r);
    LINE& operator=(LINE&& r)noexcept;
    virtual int has(int s)const;     //线路是否包含站点编号 s,返回线路中的站次序号：-1 表示没有
    virtual int cross(const LINE& b)const;   //两条公交线路相交则返回 1
    virtual operator int ()const;   //取公交线路编号
    virtual int COUNT()const;            //取公交线路的站点数量
    virtual int& operator[](int x);//取线路某个站次的站点编号
    virtual ~LINE()noexcept;
};
```

③转乘站点类：

```
class CROSS {              //从线路 from 经站点编号 stop 转至线路 to
    int begin;            //现在乘坐的公交线路
    int end;               //需要转乘的公交线路
    int stop;             //站点编号
```

```
public:
    CROSS(int begin=0, int end=0, int stop=0);
    int operator==(const CROSS& t)const;
    virtual int& getbegin();//现在乘坐的公交线路
    virtual int& getend();//需要转乘的公交线路
    virtual int& getpoint();//转乘点的站点编号
};
```

④转乘路径类：

```
class OPTION{           //一个转乘路径
    CROSS*const cross;//转乘路径上的所有转乘站点
    const int count; //转乘路径上转乘站点个数
public:
    OPTION(CROSS* tran = 0, int noft = 0);
    OPTION(const CROSS& t);
    OPTION(const OPTION& r);
    OPTION(OPTION&& r) noexcept;
    virtual int print()const;
    virtual operator int()const;            //得到转乘次数
    virtual int operator==(const OPTION& r)const;
    virtual OPTION operator *()const;       //去重复公交转乘
    virtual CROSS& operator[](int);         //取转乘
    virtual OPTION operator+(const OPTION& r)const;
    virtual OPTION& operator=(const OPTION& r);
    virtual OPTION& operator=(OPTION&& r) noexcept;
    virtual OPTION& operator+=(const OPTION& r);
    virtual ~OPTION() noexcept;
};
```

⑤转乘路径集合类：

```
class NODE {            //转乘次数和线路
    OPTION* const p; //转乘路径方案
    int n;              //转乘路径方案数
public:
    NODE(OPTION*p, int n);
    NODE(int n = 0);
    NODE(const NODE &n);
    NODE(NODE&& n)noexcept;
```

```
    virtual NODE    operator*()const; //去掉相同转乘路径
    virtual NODE    operator+(const OPTION& n)const;    //添加一条路径
    virtual NODE    operator+(const NODE& n)const;      //添加多条路径
    virtual NODE    operator*(const NODE& n)const;
    virtual NODE& operator=(const NODE& n);
    virtual NODE& operator+=(const NODE& n);
    virtual NODE& operator+=(const OPTION& n);
    virtual NODE& operator*=(const NODE& n);
    virtual NODE& operator=(NODE&& n)noexcept;
    virtual OPTION& operator [](int x);
    virtual operator int&();              //可转乘路径数 n
    virtual ~NODE()noexcept;
    virtual void print()const;          //打印转乘矩阵的元素
};
```

⑥转乘矩阵类：

```
class MAT {
    NODE *const p;   //矩阵指针
    const int r, c; //矩阵的行数和列数
public:
    MAT(int r=0,int c=0);
    MAT(const MAT& a);
    MAT(MAT&& a)noexcept;
    virtual ~MAT();
    virtual int noaccess()const;                          //若有不可达站点则返回 0
    virtual int miniTran(int b, int e, int& noft, OPTION(&r)[100])const;     //起点站次 b,终点站次 e,r 最多存 10 条路径，返回路径数
    virtual NODE* operator[](int r);                      //得到存储多种路径的 NODE 类元素的某行首址
    virtual int& operator()(int r, int c);          //得到 r 到 c 可转乘路径数目
    virtual MAT operator*(const MAT& a)const; //闭包运算：乘法
    virtual MAT operator+(const MAT& a)const; //闭包运算：加法
    virtual MAT& operator=(const MAT& a);
    virtual MAT& operator=(MAT&& a);
    virtual MAT& operator+=(const MAT& a);
    virtual MAT& operator*=(const MAT& a);
    virtual MAT& operator()(int r, int c, const OPTION& a);//将路径加入到 r 行 c 列元素中
```

```
    virtual void print()const;                          //打印转乘矩阵
};
```

⑦总括类：

```
class TOTAL {
public:
    static POINT*   point;          //所有公交站点
    static LINE*    line;           //所有公交线路
    static int    cnt_point, cnt_line;        //公交站数，公交线路数
    static MAT raw, tra;     //原始转乘矩阵，  闭包转乘矩阵
    static int    obs;              //对象数量
    TOTAL();
    TOTAL(const char* flstop, const char* flline);
    int miniTran(int fx, int fy, int tx, int ty, int& f, int& t, int& n, OPTION(&r)[100]);
    ~TOTAL();
};
```

核心函数：

⑧函数原型：int MAT::miniTran(int s, int t, int& notf, OPTION(&r)[100])const

函数功能：获取两线路之间的最少转乘次数

入口参数：起点站 s，终点站 t，最少转乘次数 notf，转乘线路数组 r

出口参数：整型变量最少转乘次数

算法思想：先遍历所有线路，寻找所有包含起点和终点的线路并将线路编号放入数组。遍历起点线路数组和终点线路数组，获取每个从起点线路到终点线路的所有转乘路线和转乘路线个数，如果线路数为 0 就继续遍历，如果不为 0 就遍历所有转乘线路，先获取转乘线路的转乘次数，如果转乘次数数组为空或者当前转乘次数小于转乘次数数组的第一个元素，就分别将转乘线路和转乘次数存入转乘线路数组和转乘次数数组的第一个元素，并让最少转乘次数线路数变量等于 1；如果当前转乘次数等于转乘次数数组的第一个元素，就将转乘线路和转乘次数存入两个数组当前的最后一个位置，并让最少转乘次数线路数变量自增，最后让 notf 等于转乘次数数组的第一个元素，并返回最少转乘次数线路数变量。

流程图：

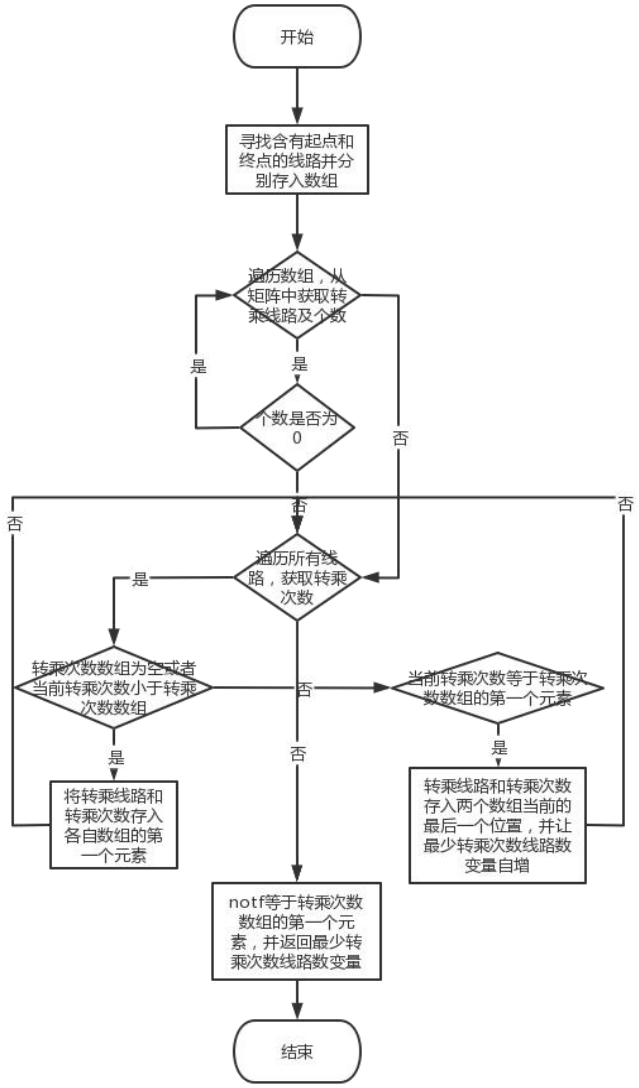


图 2-7　miniTran 函数流程图

⑨函数原型：TOTAL::TOTAL(const char* flstop, const char* flline)

函数功能：初始化总括类，特别是转乘矩阵的初始化。

入口参数：站点文件名 const char* flstop,线路文件名 const char* flline

出口参数：无

算法思想：读入站点和线路，之后构建转乘闭包矩阵，首先获取每两条线路的交点数的总和，如果交点数总和为 0，说明某条线路与其他线路之间无法转乘，抛出异常；之后构建转乘矩阵，用线路编号遍历所有线路，如果两条线路的编号相等，则某条线路内的各点可以两两相互到达，将该线路加入到矩阵中；如果编号不相等，且线路之间有交点，说明两条线路之间可以通过一次转乘抵达，这样就构建了最多一次转乘的转乘矩阵，将转乘矩阵多次相乘到线路条数-1 次，得到最终的转乘矩阵，其中储存了所有线路两两之间的所有转乘路径以及方案。

流程图：

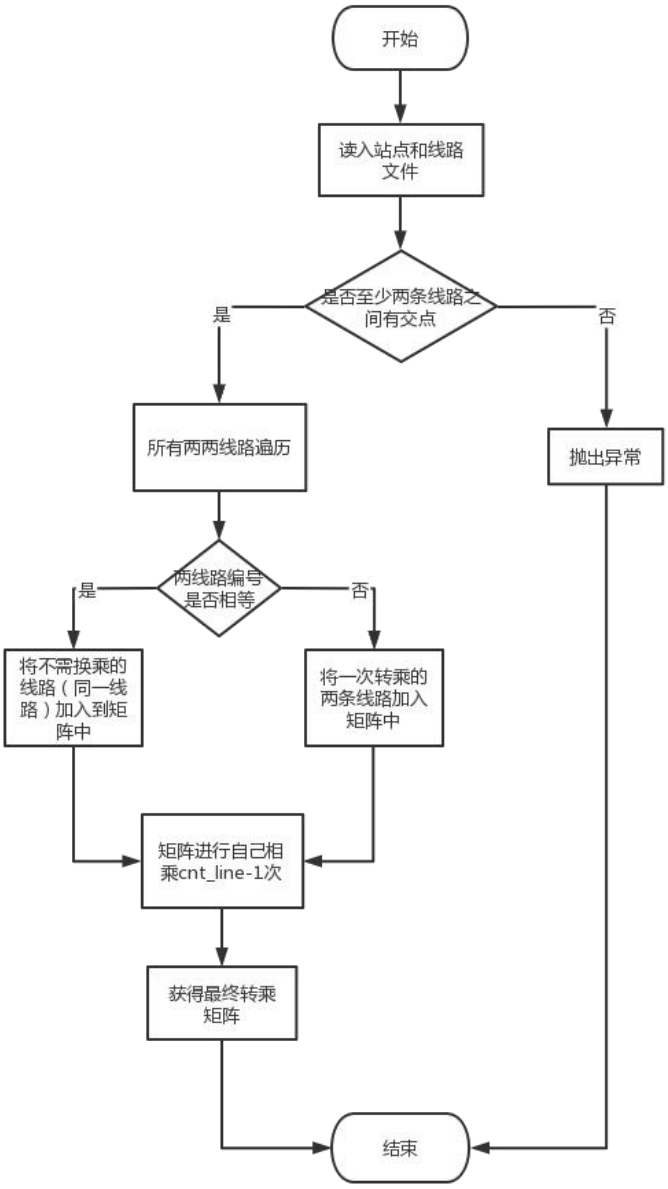


图 2-8　总括类构造函数

⑩函数原型： int TOTAL::miniTran(int fx, int fy, int tx, int ty, int& f, int& t, int& n, OPTION(&r)[100])

函数功能：根据步行起点和终点找到最少转乘次数 n 的若干线路 r，返回线路数

入口参数：起点坐标 fx，fy；终点坐标 tx，ty；起点站点编号 f，终点站点编号 t，最少转乘次数 n，线路数组 r

出口参数：线路数

算法思想：根据两点间距离公式算出与步行起点和步行终点最近的两个点，作为乘车起点和乘车终点，然后调用总括类转乘矩阵的 miniTrans 函数，获得线路个数。

流程图：

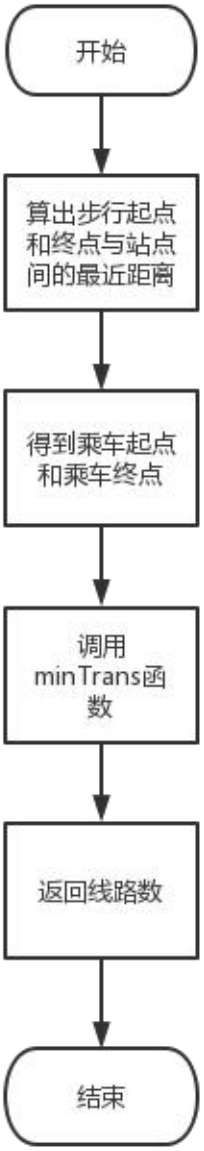


图 2-9　miniTran 函数流程图

Qt 界面模块：

①QtTipsDlgView

```
class QtTipsDlgView : public QDialog
{
    Q_OBJECT

public:
    QtTipsDlgView(const QString& msg, QWidget *parent = Q_NULLPTR);
    ~QtTipsDlgView();
```

```
    void startTimer(int);
private:
    Ui::QtTipsDlgView ui;
    QTimer* m_pTimer;
    void initFrame(const QString& msg);
};
```

功能：鼠标移动到站点的位置时显示站点信息。

槽函数：无

②QtWidgetsFL

```
class QtWidgetsFL : public QWidget
{
    Q_OBJECT

public:
    QtWidgetsFL(QWidget *parent = Q_NULLPTR);
    ~QtWidgetsFL();
    QGraphicsView* parnt;
    void myShow(QGraphicsView* p);
private:
    Ui::QtWidgetsFL ui;
private slots:
    void inputStop();
    void inputLine();
    void checkFile();
};
```

功能：站点文件和线路文件加载窗口

槽函数：inputStop(),close(),inputLine(),checkFile()

函数调用关系图：

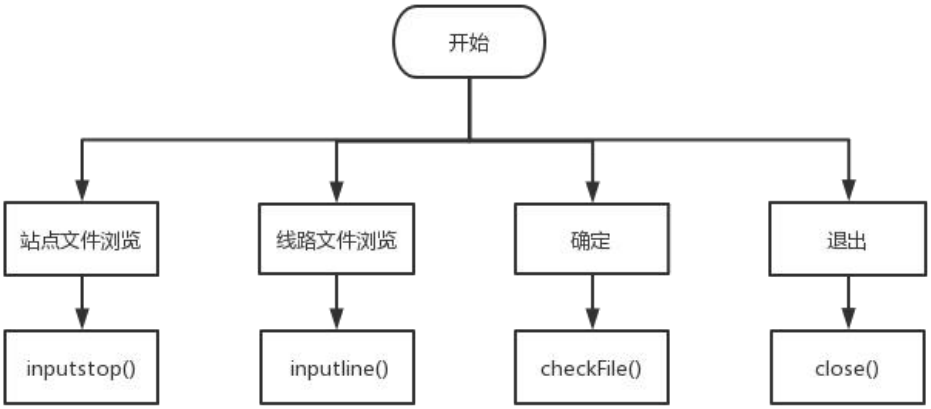


图 2-10　QtWidgetsFL 函数调用关系图

③QtWidgetsOrg

```
class QtWidgetsOrg: public QWidget
{
    Q_OBJECT

public:
    QtWidgetsOrg(QWidget *parent = Q_NULLPTR);
    ~QtWidgetsOrg();
    QGraphicsView* parnt;
    void myShow(QGraphicsView* p);

private:
    Ui::QtWidgetsOrg ui;

private slots:
    void inputOrg();
    void checkFile();
};
```

功能：地点文件加载窗口

槽函数：close(),inputOrg(),checkFile()

函数调用关系图：

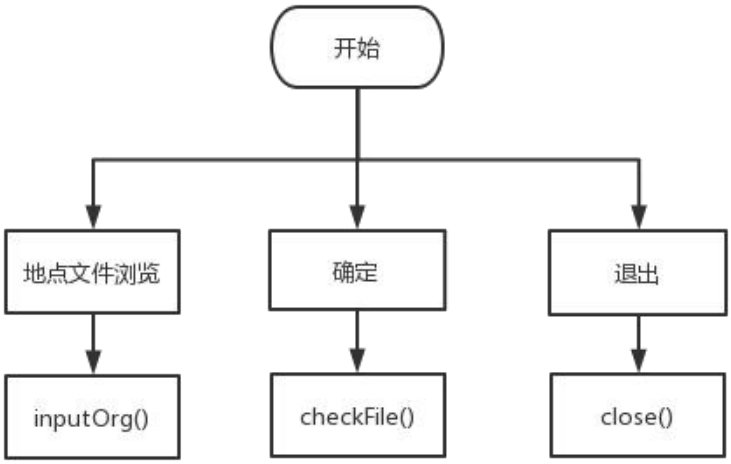


图 2-11　QtWidgetsOrg 函数调用关系图

④QtWidgetsBE

```
class QtWidgetsBE : public QWidget
{
    Q_OBJECT
public:
    QtWidgetsBE(QWidget *parent = Q_NULLPTR);
    ~QtWidgetsBE();
    QGraphicsView* parnt;
    void myShow(QGraphicsView* p);
private:
    Ui::QtWidgetsBE ui;
private slots:
    void Done();
};
```

功能：起点和终点选择窗口

槽函数：close(),Done()

函数调用关系图：

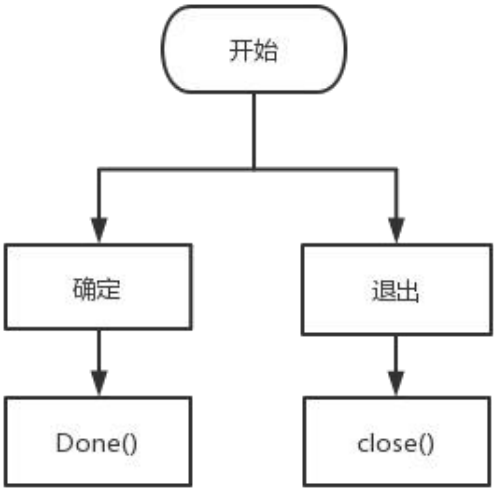


图 2-12　QtWidgetsBE 函数调用关系图

⑤map

```
class MAP : public QMainWindow
{
    Q_OBJECT
public:
    MAP(QWidget *parent = Q_NULLPTR);
    ~MAP( );
private:
    Ui::MAP ui;
    QtWidgetsFL *fl;
    QtWidgetsBE* BE;
    QtWidgetsOrg* fOrg;
    QTimer* m_Timer;
    QGraphicsItemGroup* gItem;
    void deleteItems();
protected:
    void closeEvent();
private slots:
    void loadmap();
    void closewnd();
    void mintrans();
```

```
    void loadorg();
    void btoe();
};
class MyScene : public QGraphicsScene
{
public:
    explicit MyScene(QObject* parent = 0);
    void stopLines(QGraphicsView*);
protected:
    QGraphicsView* qgv;
    void mouseMoveEvent(QGraphicsSceneMouseEvent* event);
    void mousePressEvent(QGraphicsSceneMouseEvent* event);
signals:
public slots:
};
class MyItem: public QGraphicsRectItem {
    int cx, cy;
    int cf;
    int cs;
    int bs[6];
public:
    MyItem(int x,int y, int f);
    MyItem& operator<<(int s);
    int operator()(int x, int y);
    int& x();
    int& y();
    int& f();
    int& c();
    int& operator[](int);
    int checkAllStops();
    void mousePressEvent(QGraphicsSceneMouseEvent* event);
};
```

功能：显示地图、站点、线路以及进行读取文件选择地点获取最少转乘路径等操作

槽函数：loadmap(),mintrans(),closewnd(),loadorg(),btoe()

函数调用关系图：

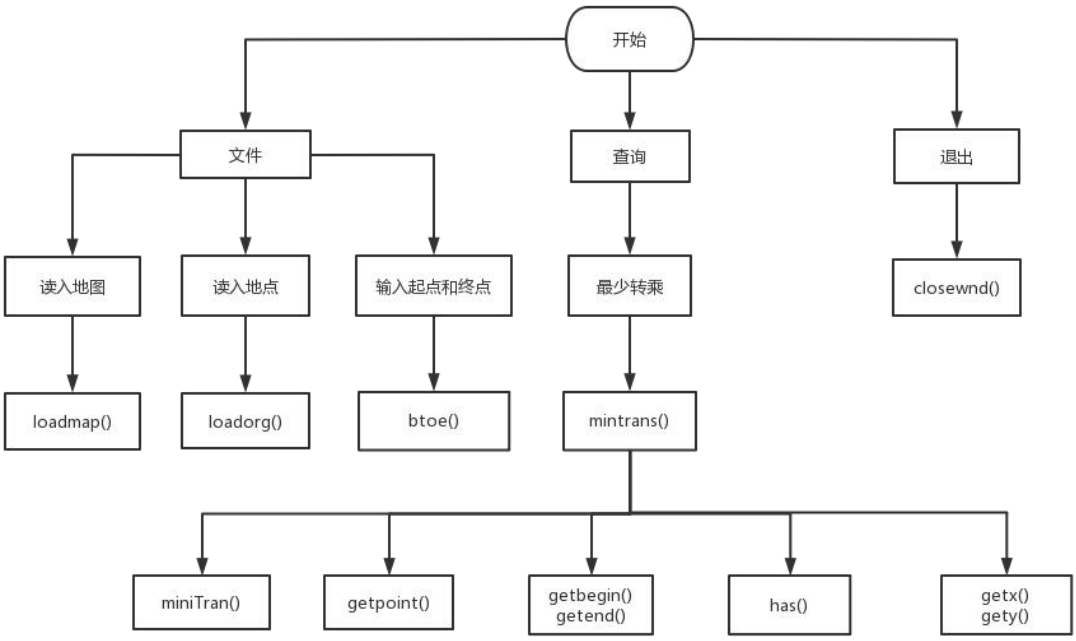


图 2-13　map 函数调用关系图

三、软件开发

本实验开发工具为 Qt 5.14.2，采用 Debug-MinGW 32bit 进行调试

四、软件测试

①主界面测试：

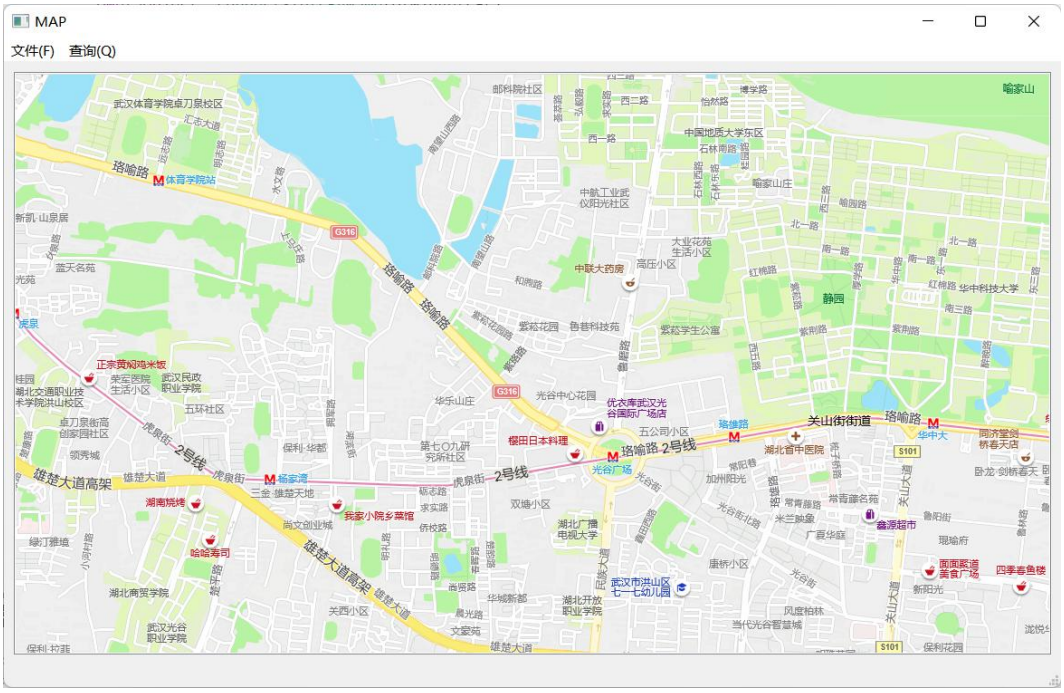


图 4-1　主界面测试

②添加地图测试：

可以输入文件路径也可以直接点击浏览读入文件路径。

图 4-2　站点及线路文件加载测试

红色的路线即为所添加的公交路线。

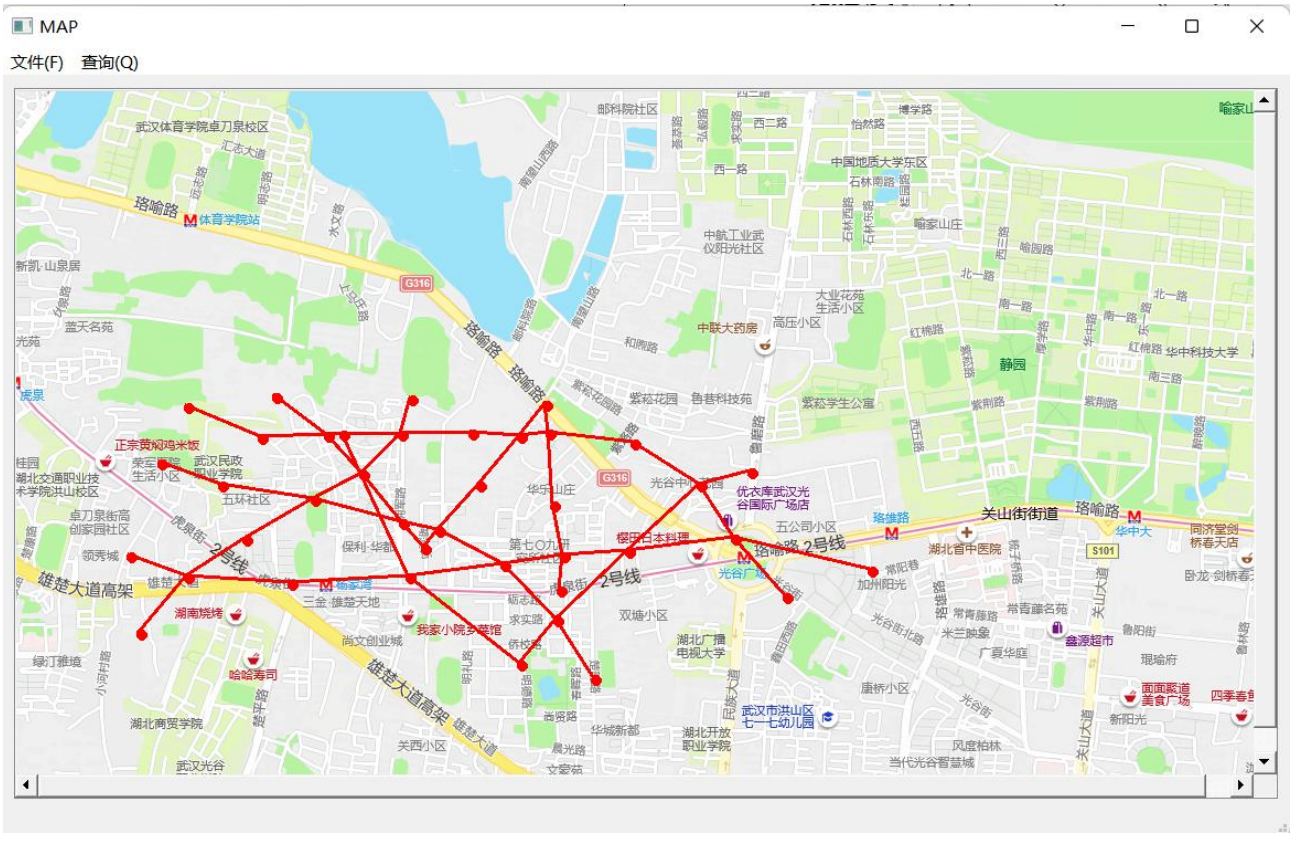


图 4-3　线路显示测试

③读入地点测试：

可以输入文件路径也可以点击浏览直接读入文件路径。

图 4-4　地点文件加载测试

④显示站点信息测试：

当鼠标移动到站点上时，会自动显示站点信息，三秒钟之后消失。

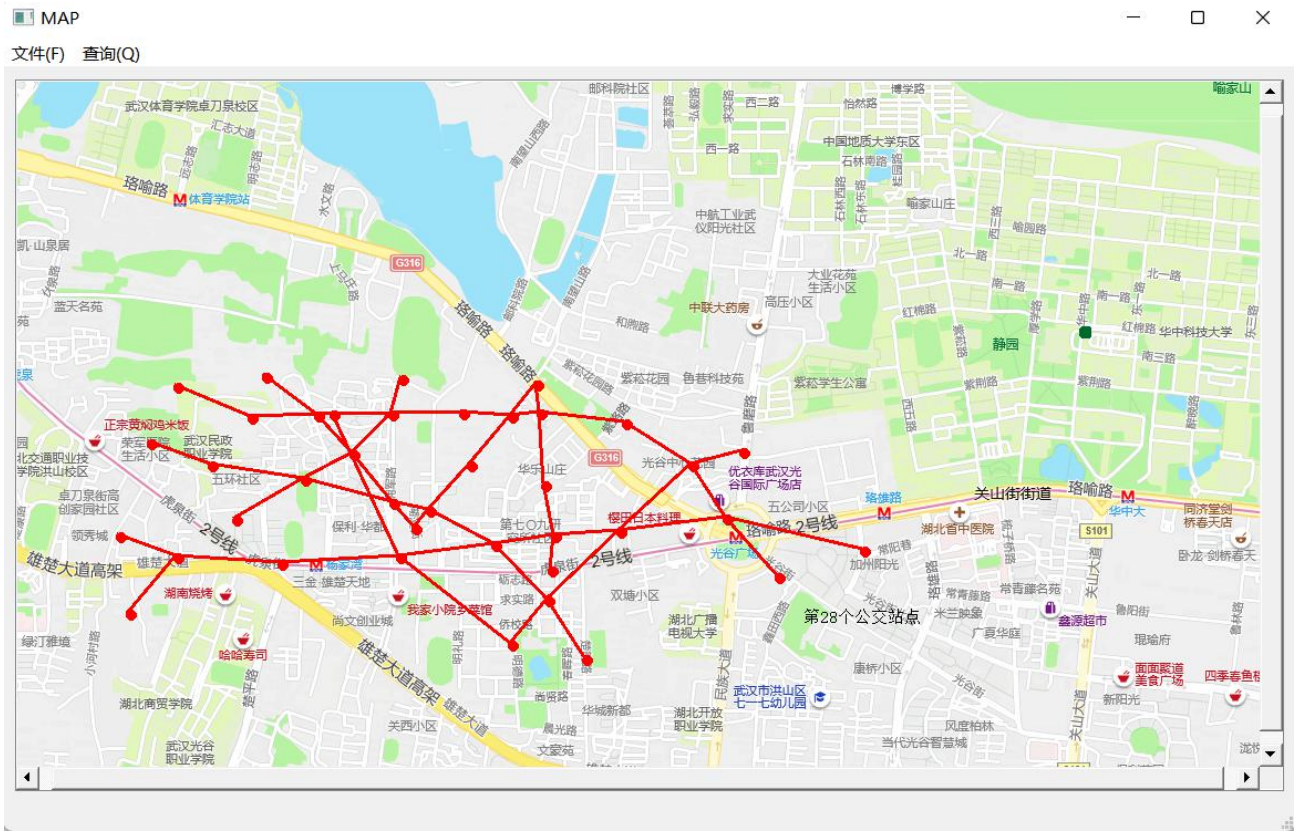


图 4-5　显示站点信息测试

⑤起点和终点选取测试：

选取起点和终点有两种方法，一种是用鼠标左键点击地图选取起点，用鼠标右键点击地

图选取终点。其中绿色点为起点，黄色点为终点。

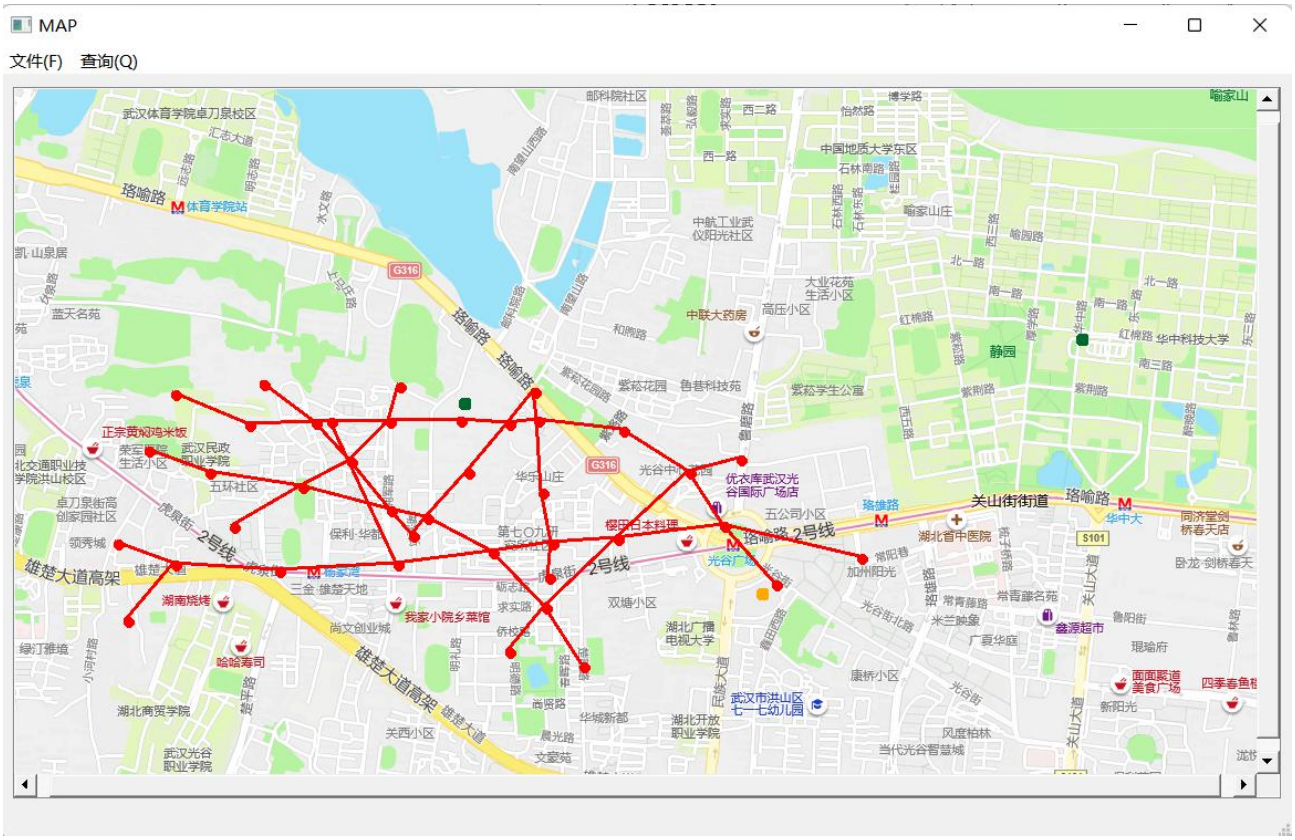


图 4-6　鼠标选取起点终点测试

另一种方法是直接在“文件”选项中选择“输入起点与终点”进行选择，用户可输入“华科大”或“华中科大”查询其所在位置，通过最大公共子串算法进行模糊匹配，找到“华中科技大学”及其坐标。

图 4-7　窗口选取起点终点测试

同样地，绿色点为起点（这里选择华中科技大学作为起点），黄色点为终点。

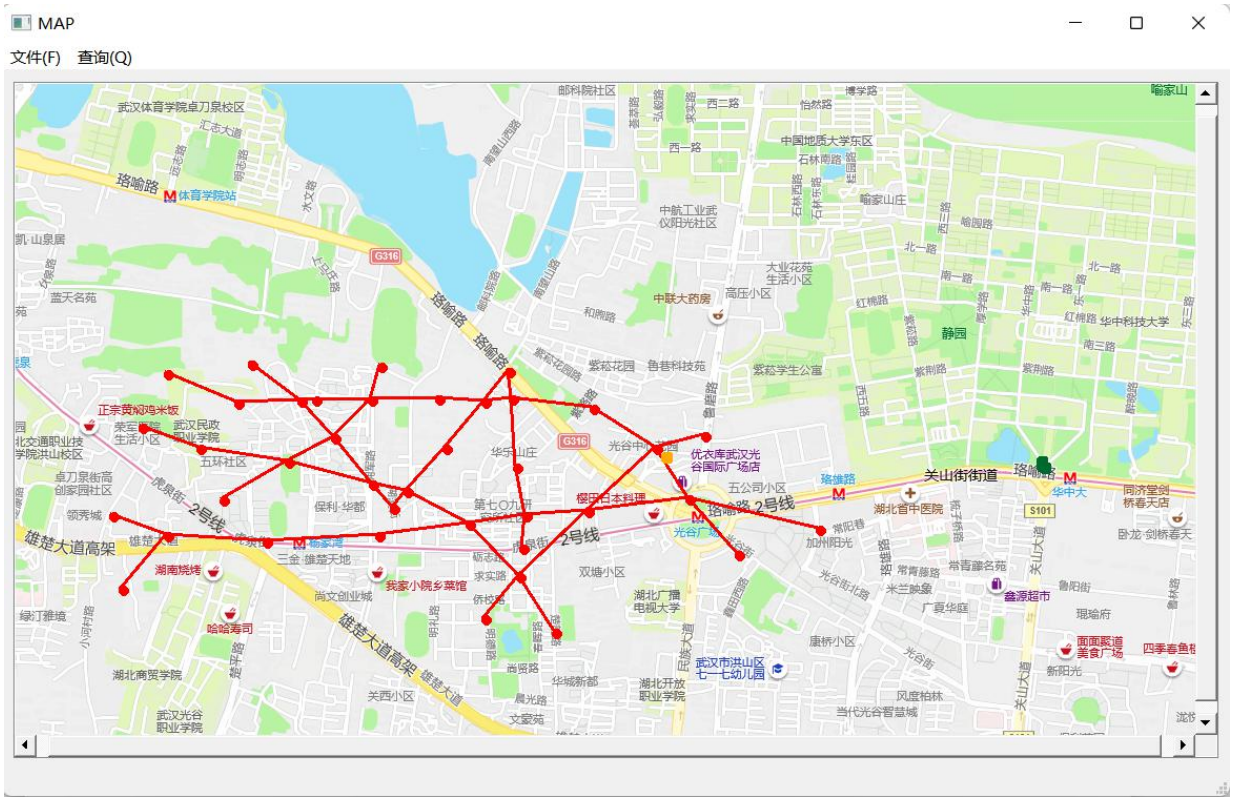


图 4-8　起点与终点选取

⑤最少转乘功能测试：

选取起点为华中科技大学，终点为光谷中心花园，选择“查询”中的“最少转乘”，可以获得最少转乘的路径，用蓝色标明。

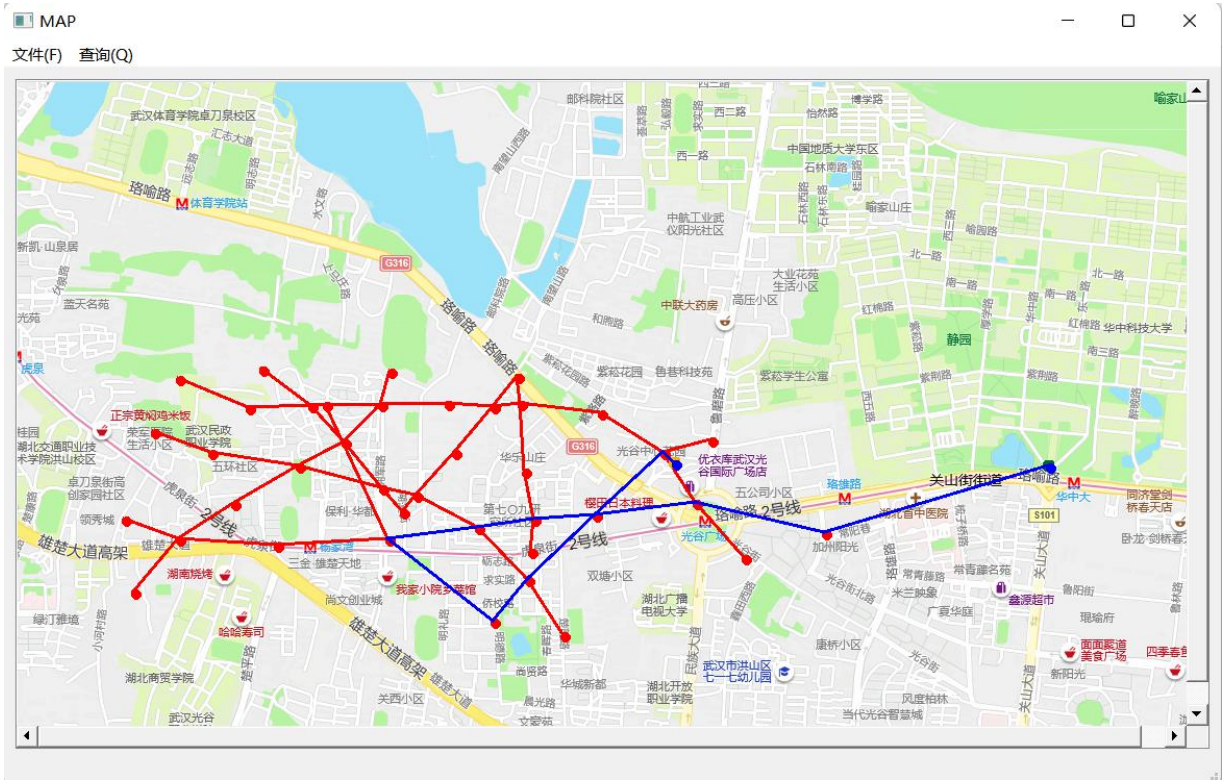


图 4-9　最少转乘功能测试

五、特点与不足

1. 技术特点

完全采用 Qt 进行代码的编写以及界面的设计，可移植性好。在选取起点和终点的时候，既可以采取鼠标点击的方式，也可以通过窗口进行选择，具有一定的灵活性和多样性

2. 不足和改进的建议

提供的地点较少，选取时有局限性，奸淫扩展地点文件。

在将地点文件读入之后，并没有在地图上的相应位置显示地点的名称，建议以后扩展此功能。

鼠标移动到站点上显示站点信息的时候，只能显示站点的编号而不能显示站点的名字，建议以后扩展此功能。

本次实验只实现了最少转乘功能，最短路径功能并没有实现，希望之后能够继续学习相关的数据结构和算法，实现该功能。

六、过程和体会

1. 遇到的主要问题和解决方法

在初次编写程序的时候，我选择了使用 vs2019+Qt 插件的模式，但当写完一部分代码并进行编译运行的时候并没有成功，去网上查阅了很多资料也没能解决问题，最终将所有代码都移植到 Qt 上，完全使用 Qt 进行编写和设计。

刚开始没能选择一个比较好的数据结构，试图采用邻接表加深度优先搜素来实现最少转乘的功能，但发现实现起来比较复杂，最后使用了老师提供的转乘矩阵结构，代码的简洁性上有了很大的改善。

在用 Qt 编译时出现了很多错误，其中绝大部分都与全局变量与静态变量在不同文件中的使用有关，在网上查阅相关资料之后将问题解决。

2. 课程设计的体会

这次的实验对我来说具有不小的挑战性，在编写之前应该将需求分析做好做仔细，否则在编写程序时会感觉迷茫而无从下手。其次应该选择适当的数据结构，编写适当的算法，不适宜的数据结构可能编写起来会相当麻烦，而且也不一定能够百分百实现所需要的功能。最后，在遇到自己不擅长的工具时，一定要细心耐心地去查阅资料学习，即使遇到很多错误也要想办法将它们全部解决，而不是仅仅让别人帮忙解决而自己不去思考。

七、源码和说明

1. 文件清单及其功能说明

bulid 开头的文件夹为编译文件夹，内含编译文件

map.pro 是 Qt 项目文件

fun.h,map.h,QtTipsDlgView.h,QtWidgetsFL.h,QtWidgetsOrg.h,QtWidgetsBE.h 为头文件

fun.cpp,map.cpp,QtTipsDlgView.cpp,QtWidgetsFL.cpp,QtWidgetsOrg.cpp,QtWidgetsBE.cpp 为源文件

map.ui,QtTipsDlgView.ui,QtWidgetsFL.ui,QtWidgetsOrg.ui,QtWidgetsBE.ui 为界面设计文件

其余为 Qt 相关配置文件

2. 用户使用说明书

方法一：用 Qt 打开 map.pro 文件，在配置好编译环境之后，直接点击左下角的第一个绿色三角形即可。

方法二：打开实验五文件夹中以 build 开头的文件夹，然后打开其中的 debug 文件夹，找到其中的.exe 文件，拖到命令行中并敲击回车，即可运行。

3. 源代码

map.pro:

```
QT       += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

CONFIG += c++11

# The following define makes your compiler emit warnings if you use
# any Qt feature that has been marked deprecated (the exact warnings
# depend on your compiler). Please consult the documentation of the
# deprecated API in order to know how to port your code away from it.
DEFINES += QT_DEPRECATED_WARNINGS \
    _GLIBCXX_USE_CXX11_ABI=0

# You can also make your code fail to compile if it uses deprecated APIs.
# In order to do so, uncomment the following line.
# You can also select to disable deprecated APIs only up to a certain version of Qt.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the APIs deprecated before Qt 6.0.0

SOURCES += \
    QtTipsDlgView.cpp \
    QtWidgetsFL.cpp \
    QtWidgetsOrg.cpp \
    QtWidgetsBE.cpp \
    fun.cpp \
    main.cpp \
    map.cpp

HEADERS += \
```

```
    QtTipsDlgView.h \
    QtWidgetsFL.h \
    QtWidgetsOrg.h \
    QtWidgetsBE.h \
    fun.h \
    map.h

FORMS += \
    QtTipsDlgView.ui \
    QtWidgetsFL.ui \
    QtWidgetsOrg.ui \
    QtWidgetsBE.ui \
    map.ui

# Default rules for deployment.
qnx: target.path = /tmp/$${TARGET}/bin
else: unix:!android: target.path = /opt/$${TARGET}/bin
!isEmpty(target.path): INSTALLS += target

RESOURCES += \
    map.qrc \
    map.qrc
```

fun.h:

```
#ifndef FUN_H
#define FUN_H
#pragma once
class POINT {              //公交站点
    int num;            //编号
    int x, y;           //坐标
public:
    POINT(int n=0, int x=0, int y=0);
    int& getx();
    int& gety();
    int& getnum();
};
class LINE{              //公交线路
    int num; //编号
    int* stop; //所有站点编号
    int count; //站点数量
public:
    LINE(int num = 0, int count = 0, int* stop = 0);
    LINE(const LINE& r);
    LINE(LINE&& r) noexcept;
    LINE& operator=(const LINE& r);
```

```
    LINE& operator=(LINE&& r)noexcept;
    virtual int has(int s)const;    //线路是否包含站点编号 s,返回线路中的站次序号：-1 表示没有
    virtual int cross(const LINE& b)const;   //两条公交线路相交则返回 1
    virtual operator int ()const;   //取公交线路编号
    virtual int COUNT()const;           //取公交线路的站点数量
    virtual int& operator[](int x);//取线路某个站次的站点编号
    virtual ~LINE()noexcept;
};
class CROSS {              //从线路 from 经站点编号 stop 转至线路 to
    int begin;           //现在乘坐的公交线路
    int end;              //需要转乘的公交线路
    int stop;           //站点编号
public:
    CROSS(int begin=0, int end=0, int stop=0);
    int operator==(const CROSS& t)const;
    virtual int& getbegin();//现在乘坐的公交线路
    virtual int& getend();//需要转乘的公交线路
    virtual int& getpoint();//转乘点的站点编号
};

class OPTION{           //一个转乘路径
    CROSS*const cross;//转乘路径上的所有转乘站点
    const int count; //转乘路径上转乘站点个数
public:
    OPTION(CROSS* tran = 0, int noft = 0);
    OPTION(const CROSS& t);
    OPTION(const OPTION& r);
    OPTION(OPTION&& r) noexcept;
    virtual int print()const;
    virtual operator int()const;            //得到转乘次数
    virtual int operator==(const OPTION& r)const;
    virtual OPTION operator *()const;      //去重复公交转乘
    virtual CROSS& operator[](int);         //取转乘
    virtual OPTION operator+(const OPTION& r)const;
    virtual OPTION& operator=(const OPTION& r);
    virtual OPTION& operator=(OPTION&& r) noexcept;
    virtual OPTION& operator+=(const OPTION& r);
    virtual ~OPTION() noexcept;
};

class NODE {            //转乘次数和线路
    OPTION* const p; //转乘路径方案
    int n;              //转乘路径方案数
public:
    NODE(OPTION*p, int n);
```

```
    NODE(int n = 0);
    NODE(const NODE &n);
    NODE(NODE&& n)noexcept;
    virtual NODE    operator*()const; //去掉相同转乘路径
    virtual NODE    operator+(const OPTION& n)const;   //添加一条路径
    virtual NODE    operator+(const NODE& n)const;      //添加多条路径
    virtual NODE    operator*(const NODE& n)const;
    virtual NODE& operator=(const NODE& n);
    virtual NODE& operator+=(const NODE& n);
    virtual NODE& operator+=(const OPTION& n);
    virtual NODE& operator*=(const NODE& n);
    virtual NODE& operator=(NODE&& n)noexcept;
    virtual OPTION& operator [](int x);
    virtual operator int&();              //可转乘路径数 n
    virtual ~NODE()noexcept;
    virtual void print()const;          //打印转乘矩阵的元素
};

class MAT {
    NODE *const p;   //矩阵指针
    const int r, c; //矩阵的行数和列数
public:
    MAT(int r=0,int c=0);
    MAT(const MAT& a);
    MAT(MAT&& a)noexcept;
    virtual ~MAT();
    virtual int noaccess()const;                        //若有不可达站点则返回 0
    virtual int miniTran(int b, int e, int& noft, OPTION(&r)[100])const;     //起点站次 b,终点站次 e,r 最多存 10 条路径，返回路径数
    virtual NODE* operator[](int r);                    //得到存储多种路径的 NODE 类元素的某行首址
    virtual int& operator()(int r, int c);          //得到 r 到 c 可转乘路径数目
    virtual MAT operator*(const MAT& a)const; //闭包运算：乘法
    virtual MAT operator+(const MAT& a)const; //闭包运算：加法
    virtual MAT& operator=(const MAT& a);
    virtual MAT& operator=(MAT&& a);
    virtual MAT& operator+=(const MAT& a);
    virtual MAT& operator*=(const MAT& a);
    virtual MAT& operator()(int r, int c, const OPTION& a);//将路径加入到 r 行 c 列元素中
    virtual void print()const;                          //打印转乘矩阵
};

class TOTAL {
public:
    static POINT*    point;          //所有公交站点
    static LINE*    line;           //所有公交线路
```

```
    static int    cnt_point, cnt_line;        //公交站数，公交线路数
    static MAT raw, tra;     //原始转乘矩阵，  闭包转乘矩阵
    static int    obs;              //对象数量
    TOTAL();
    TOTAL(const char* flstop, const char* flline);
    int miniTran(int fx, int fy, int tx, int ty, int& f, int& t, int& n, OPTION(&r)[100]);
    ~TOTAL();
};
extern TOTAL* total;
#endif // FUN_H
```

map.h:

```
#ifndef MAP_H
#define MAP_H
#pragma once
#include <QMainWindow>
#include <QApplication>
#include <QGraphicsScene>
#include <QGraphicsView>
#include <QGraphicsRectItem>
#include "ui_map.h"
QT_BEGIN_NAMESPACE
namespace Ui { class MAP; }
QT_END_NAMESPACE



class QtWidgetsFL;
class QtWidgetsOrg;
class QtWidgetsBE;
class MAP : public QMainWindow
{
    Q_OBJECT
public:
    MAP(QWidget *parent = Q_NULLPTR);
    ~MAP( );
private:
    Ui::MAP ui;
    QtWidgetsFL *fl;
    QtWidgetsBE* BE;
    QtWidgetsOrg* fOrg;
    QTimer* m_Timer;
    QGraphicsItemGroup* gItem;
    void deleteItems();
protected:
```

```
    void closeEvent();
private slots:
    void loadmap();
    void closewnd();
    void mintrans();
    void loadOrg();
    void beginEnd();
};

class MyScene : public QGraphicsScene
{

public:
    explicit MyScene(QObject* parent = 0);
    void stopLines(QGraphicsView*);
protected:
    QGraphicsView* qgv;
    void mouseMoveEvent(QGraphicsSceneMouseEvent* event);//覆盖 mousePressEvent 以捕获鼠标事件
    void mousePressEvent(QGraphicsSceneMouseEvent* event);//覆盖 mousePressEvent 以捕获鼠标事件
signals:
public slots:
};

class MyItem: public QGraphicsRectItem {
    int cx, cy;   //点击时的坐标
    int cf;         //左键点击=1，表示地点，右键点击=2 表示终点
    int cs;         //靠近该点的坐标个数
    int bs[6];    //最多存放 6 个站点的站点编号
public:
    MyItem(int x,int y, int f);
    MyItem& operator<<(int s);
    int operator()(int x, int y);
    int& x();
    int& y();
    int& f();
    int& c();
    int& operator[](int);
    int checkAllStops();
    void mousePressEvent(QGraphicsSceneMouseEvent* event);
};
#endif // MAP_H
```

QtTipsDlgView.h:

```
#ifndef QTTIPSDLGVIEW_H
#define QTTIPSDLGVIEW_H
```

```
#pragma once
#include <QTimer>
#include <QDialog>
#include "ui_QtTipsDlgView.h"

class QtTipsDlgView : public QDialog
{
    Q_OBJECT

public:
    QtTipsDlgView(const QString& msg, QWidget *parent = Q_NULLPTR);
    ~QtTipsDlgView();
    void startTimer(int);
private:
    Ui::QtTipsDlgView ui;
    QTimer* m_pTimer;
    void initFrame(const QString& msg);
};
#endif // QTTIPSDLGVIEW_H*/
```

QtWidgetsFL.h:

```
#ifndef QTWIDGETSFL_H
#define QTWIDGETSFL_H
#pragma once

#include <QWidget>
#include "map.h"
#include "ui_QtWidgetsFL.h"
#include <QApplication>
#include <QGraphicsScene>
#include <QGraphicsView>
#include <QGraphicsRectItem>

class QtWidgetsFL : public QWidget
{
    Q_OBJECT

public:
    QtWidgetsFL(QWidget *parent = Q_NULLPTR);
    ~QtWidgetsFL();
    QGraphicsView* parnt;
    void myShow(QGraphicsView* p);
private:
    Ui::QtWidgetsFL ui;
```

```
private slots:
    void inputStop();
    void inputLine();
    void checkFile();
};
#endif // QTWIDGETSFL_H
```

QtWidgetsOrg.h:

```
#ifndef QTWIDGETSOrg_H
#define QTWIDGETSOrg_H
#pragma once
#include <QWidget>
#include "ui_QtWidgetsOrg.h"
#include <string>
#include <QMainWindow>
#include <QApplication>
#include <QGraphicsScene>
#include <QGraphicsView>
#include <QGraphicsRectItem>
#include <map>
using namespace std;

class QtWidgetsOrg : public QWidget
{
    Q_OBJECT

public:
    QtWidgetsOrg(QWidget *parent = Q_NULLPTR);
    ~QtWidgetsOrg();
    QGraphicsView* parnt;
    void myShow(QGraphicsView* p);

private:
    Ui::QtWidgetsOrg ui;

private slots:
    void inputOrg();
    void checkFile();
};

void loadOrgFile(string fs);
#endif // QTWIDGETSOrg_H
```

QtWidgetsBE.h:

```
#ifndef QTWIDGETSBE_H
```

```
#define QTWIDGETSBE_H
#pragma once

#include <QWidget>
#include "ui_QtWidgetsBE.h"
#include <string>
#include <QMainWindow>
#include <QApplication>
#include <QGraphicsScene>
#include <QGraphicsView>
#include <QGraphicsRectItem>

class QtWidgetsBE : public QWidget
{
	Q_OBJECT

public:
	QtWidgetsBE(QWidget *parent = Q_NULLPTR);
	~QtWidgetsBE();
	QGraphicsView* parnt;
	void myShow(QGraphicsView* p);
private:
	Ui::QtWidgetsBE ui;

private slots:
	void Done();
};
#endif // QTWIDGETSBE_H
```

fun.cpp:

```
#define	_CRT_SECURE_NO_WARNINGS
#include "fun.h"
#include <iostream>
#include <cmath>

POINT* TOTAL::point = 0;
LINE* TOTAL::line = 0;
int TOTAL::cnt_line = 0;
int TOTAL::cnt_point = 0;
int TOTAL::obs = 0;
MAT TOTAL::raw;
MAT TOTAL::tra;

POINT::POINT(int n, int x, int y): num(n), x(x), y(y) { }
int& POINT::getx() {
```

```
    return x;
}
int& POINT::gety() {
    return y;
}
int& POINT::getnum() {
    return num;
}


LINE::LINE(int num, int count, int* stop) : num(num), stop(count ? new int[count] : 0), count(stop ? count : 0) {
    if (LINE::stop == 0 && count != 0)
        return;
    for (int x = 0; x < count; x++)
        LINE::stop[x] = stop[x];
}
LINE::LINE(const LINE& r): num(r.num), stop(r.count?new int[r.count]:0), count(stop ? r.count : 0) {
    if (stop == 0 && r.count != 0)
        return;
    for (int x = 0; x < count; x++)
        stop[x] = r.stop[x];
}
LINE::LINE(LINE&& r) noexcept: num(r.num), stop(r.stop), count(r.count) {
    (int&)(r.num) = (int&)(r.count)=0;
    (int*&)(r.stop) = 0;
}
LINE& LINE::operator=(const LINE& r) {
    if (this == &r) return *this;
    if (stop) { delete[] stop; }
    (int&)num = r.num;
    (int*&)stop = new int[r.count];
    (int&)count = stop ? r.count : 0;
    for (int x = 0; x < count; x++)
        stop[x] = r.stop[x];
    return *this;
}
LINE& LINE::operator=(LINE&& r)noexcept{
    if (this == &r) return *this;
    if (stop) { delete[] stop; }
    (int&)num = r.num;
    (int*&)stop = r.stop;
    (int&)count = r.count;
    (int&)(r.num) = (int&)(r.count) = 0;
    (int*&)(r.stop) = 0;
    return *this;
```

```
}
int LINE::has(int s)const {
	for (int x = 0; x < count; x++)
		if (stop[x] == s) return x;
	return -1;
}
int LINE::cross(const LINE& b)const {	//线路相交则返回 1
	if (this == &b) return 0;
	for (int x = 0; x < count; x++)
		if (b.has(stop[x]) != -1) return 1;
	return 0;
}
LINE::operator	int () const{
	return num;
}		//取线路编号
int LINE::COUNT()const {
	return count;
}			//取站点数目
int& LINE::operator[](int x) {
	return stop[x];
}	//第 x 站的站点编号
LINE::~LINE()noexcept {
	if (stop) {
		delete[] stop;
		(int*&)stop = 0;
		(int&)num = (int&)count = 0;
	}
}


CROSS::CROSS(int begin, int end, int stop): begin(begin), end(end), stop(stop) { }
int CROSS::operator==(const CROSS& t)const {
	return begin == t.begin && end == t.end && stop == t.stop;
}
int& CROSS::getbegin() {
	return begin;
}
int& CROSS::getend() {
	return end;
}
int& CROSS::getpoint() {
	return stop;
}
```

```
OPTION::OPTION(CROSS* tran, int noft): cross(noft?new CROSS[noft]:0), count(tran ? noft : 0) {
    if (cross == 0 && noft!=0)
        return;
    for (int x = 0; x < noft; x++)
        cross[x] = tran[x];
}
OPTION::OPTION(const CROSS& t): cross(new CROSS[1]), count(cross ? 1 : 0) {
    if (cross == 0)
        return;
    *cross = t;
}
OPTION::OPTION(const OPTION& r): cross(r.count?new CROSS[r.count]:0), count(cross ? r.count : 0) {
    if (cross == 0 && r.count!=0)
        return;
    for (int x = 0; x < count; x++)
        cross[x] = r.cross[x];
}
OPTION::OPTION(OPTION&& r) noexcept : cross(r.cross), count(r.count) {
    (CROSS*&)(r.cross) = 0;
    (int&)(r.count)=0;
}
OPTION::~OPTION() noexcept {
    if (cross) {
        delete[]cross;
        (CROSS*&)cross = 0;
        *(int*)&count = 0;
    }
}
int OPTION::print()const {
    for (int x = 0; x < count; x++)
        if (cross[x].getpoint() == -1)
            printf("\tstay on line %d and go directly\n", cross[x].getbegin() + 1);
        else
            printf("\tfrom line %d to line %d via %d\n", cross[x].getbegin() + 1, cross[x].getend() + 1, cross[x].getpoint() +
1);
    return count;
}
OPTION::operator int()const {
    return count;
}


OPTION OPTION::operator *()const { //去重复转乘
    int cnt = count;
    CROSS* t = new CROSS[count];
```

```
    for (int x = 0; x < cnt; x++) t[x] = cross[x];
    for(int x=0; x<cnt-1; x++)
        for (int y = x + 1; y < cnt; y++) {
            if (t[x].getpoint() == t[y].getpoint()&&t[x].getbegin() == t[y].getend()) { //循环了
                for (int m=x,n=y+1; n <cnt; n++, m++)
                    t[m] = t[n];
                cnt -= (y+1 - x);
                y = x;
            }
        }
    OPTION r(t, cnt);
    delete[]t;
    return r;
}
int OPTION::operator==(const OPTION& r)const {
    int m = 1;
    if (count != r.count)
        return 0;
    for (int x = 0; x < count; x++) {
        if (cross[x] == r.cross[x])
            continue;
        m = 0;
        break;
    }
    return m;
}
CROSS& OPTION::operator[](int x) { //一条线路上所有转乘
    if (x<0 || x>count)
        throw"error";
    return cross[x];
}
OPTION    OPTION::operator+(const OPTION& r)const { //两段路径连接
    int x, y;
    OPTION s;
    if (count == 0) return *this;
    if (r.count == 0) return r;
    if (cross[count - 1].getend() != r.cross[0].getbegin())
        return *this;
    (CROSS*&)(s.cross) = new CROSS[count + r.count];
    (int&)(s.count) = s.cross ? count + r.count : 0;
    for (x = 0; x < count; x++) s.cross[x] = cross[x];
    for (y = 0; y < r.count; y++) s.cross[x++] = r.cross[y];
    return *s;
}
OPTION& OPTION::operator=(const OPTION& r) {
```

```
    if (this == &r) return *this;
    if (cross) delete[]cross;
    (CROSS*&)cross = new CROSS[r.count];
    (int&)count = cross ? r.count : 0;
    for (int x = 0; x < count; x++) cross[x] = r.cross[x];
    return *this;
}
OPTION& OPTION::operator=(OPTION&& r) noexcept{
    if (this == &r) return *this;
    if (cross) delete[]cross;
    (CROSS*&)cross =r.cross;
    (int&)count = r.count;
    (CROSS*&)(r.cross) = nullptr;
    (int&)(r.count) = 0;
    return *this;
}
OPTION& OPTION::operator+=(const OPTION& r) {
    return *this = *this + r;
}



NODE::NODE(OPTION* p, int n):p(n?new OPTION[n]:0), n(p?n:0) {
    if (this->p == 0 && n!=0)
        return;
    for (int x = 0; x < n; x++) this->p[x] = p[x];
}
NODE::NODE(int n):p(n?new OPTION[n]:0), n(p?n:0) {
    if (p == 0 && n!=0)
        return;
}
NODE::NODE(const NODE& n) : p(n.n ? new OPTION[n.n] :0), n(p ? n.n : 0) {
    if (p == 0 && n.n!=0)
        return;
    for (int x = 0; x<NODE::n; x++) p[x] = n.p[x];
}
NODE::NODE(NODE&& n)noexcept: p(n.p), n(n.n) {
    (OPTION*&)(n.p) = 0;
    (int&)n.n = 0;
}
NODE    NODE::operator*()const {
    int n = this->n;
    if (n == 0) return *this;
    OPTION* t = new OPTION[n];
```

```
    for (int x = 0; x < n; x++) t[x] = p[x];
    for (int x = 0; x < n - 1; x++)
        for (int y = x + 1; y < n; y++) {
            if (t[x]== t[y]) {
                for (int m = x + 1, n = y + 1; n < n - 1; n++, m++)
                    t[m] = t[n];
                n -= (y - x);
                y = x;
            }
        }
    NODE r(t, n);
    //try {
    //    if (t != nullptr)delete[]t;
    //    t = nullptr;
    //}
    //catch (...) {
    //    throw "initializing failed！";
    //}
    return r;
}
NODE NODE::operator+(const OPTION& n)const {
    NODE r(this->n + 1);
    for (int x = 0; x < this->n; x++) r.p[x] = *p[x];
    r.p[this->n] = n;
    return *r;
}
NODE NODE::operator+(const NODE& n)const {
    if (this->n == 0) return n;
    if (n.n == 0) return *this;
    NODE r(this->n + n.n);
    for (int x = 0; x < this->n; x++) r.p[x] = *p[x];
    for (int x = 0; x < n.n; x++) r.p[x+this->n] = *n.p[x];
    return *r;
}
NODE NODE::operator*(const NODE& n)const {
    if (this->n == 0) return *this;
    if (n.n == 0) return n;
    NODE r(this->n * n.n);
    int k = 0;
    for(int x=0; x< this->n; x++)
        for (int y = 0; y < n.n; y++) {
            if (p[x][p[x]-1].getend() != n.p[y][0].getbegin()) throw "Can not tansship from buses!";
            try {
                r.p[k] = p[x] + n.p[y];   //连接运算
            }
```

```
                catch (const char* e) {
                    //const char* p = e;
                }
                k++;
            }
        return *r;
}
NODE& NODE::operator=(const NODE& n) {
    if(this == &n) return *this;
    if (p) delete[]p;
    (OPTION*&)p = new OPTION[n.n];
    (int&)(NODE::n) = p ? n.n : 0;
    for (int x= 0; x < n.n; x++) p[x] = n.p[x];
    return *this;
}
NODE& NODE::operator=(NODE&& n)noexcept {
    if (this == &n) return *this;
    if (p) delete[]p;
    (OPTION*&)p = n.p;
    (int&)(NODE::n) = n.n;
    (OPTION*&)(n.p) = nullptr;
    (int&)(n.n) = 0;
    return *this;
}
NODE& NODE::operator+=(const OPTION& n) {
    return *this = *this + n;
}
NODE& NODE::operator+=(const NODE & n)    {
    return *this = *this + n;
}
NODE& NODE::operator*=(const NODE& n) {
    return *this = *this * n;
}
OPTION& NODE::operator[](int x) {
    if (x < 0 || x >= n)
        throw"error";
    return p[x];
}
NODE::operator int&() {
    return n;
}
NODE::~NODE()noexcept {
    if (p) {
        delete[]p;
        (OPTION*&)p = 0;
```

```
            (int&)n = 0;
        }
}
void NODE::print()const {
        for (int m = 0; m < n; m++) {
            p[m].print();
        }
}



MAT::MAT(int r, int c) : p((r!=0&&c!=0)?new NODE [r * c]:0),    r(p ? r : 0), c(p? c : 0) {
        if (MAT::p == 0 && r != 0 && c != 0) return;
}
MAT::MAT(const MAT& a) : p((a.r!=0 && a.c!=0)?new NODE [a.r*a.c]:0), r(p? a.r:0), c(p ? a.c:0) {
        if (p == 0 && a.r != 0 && a.c != 0) return;
        for (int k = 0; k < r * c - 1; k++) p[k] = a.p[k];
}
MAT::MAT(MAT&& a)noexcept: p(a.p), r(a.r), c(a.c){
        (NODE*&)(a.p) = 0;
        (int&)(a.r) = (int&)(a.c) = 0;
}
MAT::~MAT() {
        if (p) {
            delete[] p;
            (NODE*&)p = 0;
            (int&)r = (int&)c = 0;
        }
}
int MAT::noaccess()const {
        for (int x = r * c - 1; x >= 0; x--) if (p[x].operator int &()==0) return 0;
        return 1;
}
int& MAT::operator()(int x, int y) {
        if (x < 0 || x >= r) throw"error";
        if (y < 0 || y >= c) throw"error";
        return p[x * c + y];
}
NODE* MAT::operator[](int r) {
        if (r < 0 || r >= MAT::r) throw"error";
        return p+r * c;
}
MAT& MAT::operator=(const MAT& a) {
        if (this == &a) return *this;
        if (p) delete[]p;
```

```
    (NODE*&)p = new NODE[a.r*a.c];
    (int&)r = a.r;
    (int&)c = a.c;
    for (int k = r * c - 1; k >= 0; k--)
        p[k] = a.p[k];
    return *this;
}
MAT& MAT::operator=(MAT&& a) {
    if (this == &a) return *this;
    if (p) delete[]p;
    (NODE*&)p = a.p;
    (int&)r = a.r;
    (int&)c = a.c;
    (NODE*&)(a.p) = 0;
    (int&)(a.r)=(int&)(a.c)=0;
    return *this;
}
MAT MAT::operator*(const MAT& a)const {
    if (c != a.r)
        throw"error";
    int t;
    MAT s(r, a.c);                          //每个节点皆为空线路
    for (int h = 0; h < r; h++)
        for (int j = a.c - 1; j >= 0; j--) {
            if (h == j) continue;
            t = h * s.c + j;
            for (int k = 0; k < c; k++)
                if (k != h && k!=j) //新增路线数
                    s.p[t]+= p[h * c + k] * a.p[k * a.c + j];
        }
    return s;
}
MAT MAT::operator+(const MAT& a)const {
    if (r!=a.r || c !=a.c )
        throw"error";
    MAT s(*this);
    for (int h = r * c - 1; h >= 0; h--)
        s.p[h] += a.p[h];
    return s;
}
MAT& MAT::operator+=(const MAT& a) {
    return *this = *this + a;
}
MAT& MAT::operator*=(const MAT& a) {
    return *this = *this * a;
```

```
}
MAT& MAT::operator()(int ro, int co, const OPTION& a) {
    p[ro * c + co] += a;
    return *this;
}
//根据起点站 s 和终点站 t 找到最少转乘次数 noft 的若干线路 r,返回线路数
int MAT::miniTran(int s, int t, int& notf, OPTION(&r)[100])const {
    int u, v, w, x, y, z;//z:返回实际最少转乘线路数
    int b = 0, e = 0;           //包含起点 s 的起始线路数 b(线路存放在 bls),包含终点 t 的起始线路数 e(线路存放在 els)
    int nott[100]{};            //对应规划线路 r 的转乘次数
    int bls[20], els[20];     //bls:包含起点 s 的起始线路
    NODE rou;
    for (z = 0; z < TOTAL::cnt_line; z++) { //寻找包含起点或终点相关公交线路下标
        if (TOTAL::line[z].has(s) != -1)
            if (b < 20)
                bls[b++] = z;
        if (TOTAL::line[z].has(t) != -1)
            if (e < 20)
                els[e++] = z;
    }
    for (x = z = 0; x < b; x++)
        for (y = 0; y < e; y++) {
            rou = TOTAL::tra[bls[x]][els[y]]; //得到两线路的所有转乘线路
            w = TOTAL::tra(bls[x], els[y]);
            if (w == 0)
                continue;     //转乘线路数==0
            for (v = 0; v < w; v++) {
                u = rou[v];             //线路得到转乘次数
                if (z == 0 || u < nott[0]) {//若 nott 为空，或转乘次数比 nott[0]小
                    nott[0] = u;
                    r[0] = rou[v];
                    z = 1;
                }
                if (u == nott[0]) { //和已有线路转乘次数相同时，则插入
                    if (z == 100) return z;
                    nott[z] = u;
                    r[z] = rou[v];
                    z++;
                }
            }
        }
    notf = nott[0]; //nott[0]到的 nott[z-1]的转乘次数都相同
    return z;           //返回最少转乘线路数
}
void MAT::print()const {
```

```
    for(int x=0; x < r; x++)
        for (int y = 0; y < c; y++) {
            printf("Node(%d,%d) has %d routes:\n", x, y, (int)(p[x * c + y]));
            p[x * c + y].print();
        }

}


TOTAL::TOTAL() {
    obs++;
}
TOTAL::TOTAL(const char* flstop, const char* flline) {
    int    m, n, p, q, r, * s, * t;
    FILE* fs, * fl;
    fs = fopen(flstop, "r");
    fl = fopen(flline, "r");
    if (fs == 0 || fl == 0) throw "File open error!";
    fscanf(fs, "%d", &cnt_point);
    point = new POINT[cnt_point];
    for (m = 0; m < cnt_point; m++) {
        fscanf(fs, "%d%d", &point[m].getx(), &point[m].gety());
        point[m].getnum() = m + 1;          //公交线路编号从 1 开始
    }
    fclose(fs);
    fscanf(fl, "%d", &cnt_line);
    s = new int[cnt_line];                  //每条线路的站点数
    t = new int[100];                   //每条线路的站点数不超过 100 站
    for (m = 0; m < cnt_line; m++) {
        fscanf(fl, "%d", &s[m]);
    }
    *(LINE**)&line = new LINE[cnt_line];
    for (m = 0; m < cnt_line; m++) {
        for (n = 0; n < s[m]; n++) {
            fscanf(fl, "%d", &t[n]);
            t[n]--;
        }
        line[m] = LINE(m + 1, s[m], t);
    }
    fclose(fl);
    //判断是否有交点，没有交点就无法构建闭包矩阵
    for (m = 0; m < cnt_line; m++) {
        for (p = n = 0; n < cnt_line; n++)
            if (m != n) p += TOTAL::line[m].cross(TOTAL::line[n]);
        if (p == 0) {
```

```
                printf("line %d does nott cross any line\n", m + 1);
                throw    "there is independent line";
            }
        }
        MAT ra(cnt_line, cnt_line);
        OPTION a;
        CROSS* u = new CROSS[100];
        for (m = 0; m < cnt_line; m++)
            for (n = 0; n < cnt_line; n++)
            {
                if (m == n) {                           //某条线路内的各点可以两两相互到达
                    u[0] = CROSS(m, n, -1);
                    a = OPTION(&u[0],1 );
                    ra(m, n, a);
                    continue;
                }
                p = 0;                                  //某两条线路的交点个数
                for (q = TOTAL::line[m].COUNT() - 1; q >= 0; q--) {
                    r = TOTAL::line[m][q];
                    if (TOTAL::line[n].has(r) != -1)    //两条线路有交点，可以通过一次换乘到达
                    {
                        u[p] = CROSS(m, n, r);
                        a = OPTION(&u[p++],1); //线路只有一次转乘
                        ra(m, n, a);
                    }
                }
            }
        tra = raw = ra;
        for (n = 2; n < cnt_line; n++) {//构造闭包矩阵
            raw *= ra;
            tra += raw;
        }
        raw = ra;
        delete s;
        delete t;
        delete[]u;
        obs++;
    }
    TOTAL::~TOTAL() {
        obs--;
        if (obs) return;
        if (point) {
            delete[]point;
            *(POINT**)&point = 0;
        }
```

```
    if (line) {
        delete[]line;
        *(LINE**)&line = 0;
    }
}
//根据步行起点和终点找到最少转乘次数 n 的若干线路 r，返回线路数
int TOTAL::miniTran(int fx, int fy, int tx, int ty, int& f, int& t, int& n, OPTION(&r)[100]) {
    int m;
    double df, tf, dt, tt;
    f = 0;          //设离起点最近的站点 f
    df = sqrt((point[0].getx() - fx) * (point[0].getx() - fx) + (point[0].gety() - fy) * (point[0].gety() - fy));
    t = 0;          //设离终点最近的站点 t
    dt = sqrt((point[0].getx() - tx) * (point[0].getx() - tx) + (point[0].gety() - ty) * (point[0].gety() - ty));
    for (m = 1; m < TOTAL::cnt_point; m++) {                //搜索离起点或终点最近的站点
        tf = sqrt((point[m].getx() - fx) * (point[m].getx() - fx) + (point[m].gety() - fy) * (point[m].gety() - fy));
        if (df > tf) { df = tf;   f = m; }    //离步行起点最近的站点，存在 f 中
        tt = sqrt((point[m].getx() - tx) * (point[m].getx() - tx) + (point[m].gety() - ty) * (point[m].gety() - ty));
        if (dt > tt) { dt = tt;   t = m; }
    }
    if (f == t) return 0;    //起点和终点相同，不用乘车
    return TOTAL::tra.miniTran(f, t, n, r);
}
```

main.cpp:

```
#include "map.h"
#include <QtWidgets/QApplication>
#include <qdesktopwidget.h>
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MAP w;
    w.show();
    return a.exec();
}
```

map.cpp:

```
#include "QtTipsDlgView.h"
#include "map.h"
#include "ui_map.h"
#include "fun.h"
#include "QtWidgetsFL.h"
#include "QtWidgetsOrg.h"
#include "QtWidgetsBE.h"
#include <cmath>
#include <QDialog>
```

```
#include <QDesktopWidget>
#include <QApplication>
#include <QMessageBox>
#include <QGraphicsSceneMouseEvent>
#include <QGraphicsEllipseItem>
TOTAL *total=0;
bool route=false;            //未显示查询路径时
bool depart = false;       //未选取步行起点
bool arrive = false;       //未选取步行终点
MyItem::MyItem(int x, int y, int f): QGraphicsRectItem(x-3,y-3,7,7), cx(x), cy(y), cf(f), cs(0) {
    //以下根据鼠标左键点击步行起点和鼠标右键点击步行终点设置不同画笔颜色
    QBrush qbrush(f==1? QColor(00, 105, 45) : QColor(255, 170, 00));     //根据 f 设置颜色
    QPen qpens(qbrush, 4, Qt::SolidLine, Qt::SBEareCap, Qt::BevelJoin); //图元点的画笔
    QGraphicsRectItem* item = this;
    item->setPen(qpens);
    item->setBrush(qbrush);
    item->setFlag(QGraphicsItem::ItemIsMovable); //不选中：因为没有移动图元点的操作
    checkAllStops(); //检测所有站点，找出最近的站点存于 bs 中
}
int& MyItem::x() { return cx; }
int& MyItem::y() { return cy; }
int& MyItem::f() { return cf; }
int& MyItem::c() { return cs; }
int& MyItem::operator[](int x) {
    if (x < 0 || x >= cs) throw "subscription overflow for stops around departure point!";
    return bs[x];     //返回已检测的编号为 x 的站点编号
}
int MyItem::operator()(int x, int y) {   //仅用于检测两点距离远近，不开方
    return (x - cx) * (x - cx) + (y - cy) * (y - cy);
}
MyItem& MyItem::operator<<(int s) {
    if (s < 0 || s >= TOTAL::cnt_point) return *this;
    int d = (*this)(TOTAL::point[s].getx(), TOTAL::point[s].gety()); //d 为当前图元点到站点 s 的距离
    int m;                  //m 为当前图元点到先前已检测站点的距离
    if(cs==0||d<(m = (*this)(TOTAL::point[bs[0]].getx(), TOTAL::point[bs[0]].gety()))){ //若 bs 没有元素即 cs==0，或距离站点 s 更近
        bs[0] = s;
        cs = 1;
        return *this;
    }
    if (d == m) {     //和已检测站点比，到标号为 s 的站点的距离相同时
        if (cs == 6) return *this;
        bs[cs] = s; //只保存和最近距离相同的站点
        cs++;            //距离相同的站点个数增加
    }
```

```
        return *this;
    }
    int MyItem::checkAllStops(){//检测所有站点，找出最近的站点存于 bs 中
        for(int c=0;c<TOTAL::cnt_point; c++)
            operator<<(TOTAL::point[c].getnum());
        return cs;                    //返回距离最近且相同的站点个数
    }
    void MyItem::mousePressEvent(QGraphicsSceneMouseEvent* event) {
        setSelected(true);          //当前图元被选中
        QGraphicsRectItem::mousePressEvent(event);
    }
    //定义自己的场景 MyScene，以便能够捕获鼠标或键盘事件
    MyScene::MyScene(QObject* parent): QGraphicsScene(parent)
    {
        clearFocus();
        qgv = Q_NULLPTR;              //没有加载地图文件时
    }
    void MyScene::mouseMoveEvent(QGraphicsSceneMouseEvent* event) {
        //注意在其.ui 界面文件中，mouseTracking 必须勾选，否则不会出现此事件
        if (qgv == Q_NULLPTR) { //如果没有加载地图文件
            QGraphicsScene::mouseMoveEvent(event);
            return;
        }
        QPointF qpointf = event->scenePos();          //获取鼠标移动时的坐标
        for (int n = 0; n < total->cnt_point; n++) {
            if (fabs(total->point[n].getx() - qpointf.x()) < 8 && fabs(total->point[n].gety() - qpointf.y()) < 8) {
                //以下提示信息必须使用 fromOrgal8Bit()，否则中文提示会显示乱码
                QtTipsDlgView dlg(QString::fromUtf8("第")+QString::number(n+1, 10, 0)+QString::fromUtf8("个公交站点
"));
                //dlg.setAttribute(Qt::WA_ShowModal,true); 若调用 show()则需设置无边框,若调用 dlg.exec()则不用此行，
                dlg.startTimer(2000);                 //设置悬停显示时间为 2 秒，时间到自动关闭 dlg
                QPointF m1 = qgv->mapToGlobal(QPoint(qpointf.x(), qpointf.y()));
                dlg.move(QPoint(m1.x(), m1.y()));
                dlg.exec();                           //显示站点提示信息
            }
        }
        QGraphicsScene::mouseMoveEvent(event);    //回调基类鼠标事件
    }
    void MyScene::mousePressEvent(QGraphicsSceneMouseEvent* event)
    {
        if (qgv == Q_NULLPTR||    route==true) {        //未加载地图及显示查询结果时，不响应鼠标按下事件
            QGraphicsScene::mouseMoveEvent(event);
            return;
        }
        QPointF qpointf = event->scenePos();          //获取鼠标坐标
```

```
    QList<QGraphicsItem*> listItem = items();
    int lb = 0;
    if (event->button() == Qt::LeftButton) { lb = 1; depart = true; }     //检查左键是否按下
    if (event->button() == Qt::RightButton) { lb = 2; arrive = true; }    //检查左键是否按下
    for (int i = listItem.length() - 1; i >= 0; i--) {
        MyItem* item = (MyItem*)listItem[i];
        if (item->f() == lb) {
            listItem.removeAt(i);
            delete item;
            break;
        }
    }
    addItem(new MyItem(qpointf.x(), qpointf.y(), lb));
    QGraphicsScene::mousePressEvent(event);                    //回调基类鼠标事件
}
void MyScene::stopLines(QGraphicsView* parnt) {        //加载地图站点和线路
        //按视图 graphicsview 大小设置 scene 显示区域大大小
        QSize viewsize = parnt->size();                        //取得 graphicsview 视图区域大小
        MyScene* scene;
        if (parnt->scene() != Q_NULLPTR)     delete parnt->scene();
        scene = new MyScene(parnt);                            //创建 scene
        scene->setSceneRect(0, 0, viewsize.width(), viewsize.height());
        scene->qgv = parnt;
        //显示所有公交站点
        QBrush qbrush(QColor(255, 0, 0));                  //设置颜色
        QPen qpens(qbrush, 4, Qt::SolidLine, Qt::SBEareCap, Qt::BevelJoin); //站点的笔
        for (int n = 0; n < total->cnt_point; n++){
            scene->addEllipse(total->point[n].getx(), total->point[n].gety(), 6, 6, qpens, qbrush);
        }
        //显示所有线路
        QPen qpenl(qbrush, 3, Qt::SolidLine, Qt::SBEareCap, Qt::BevelJoin); //线路的笔
        for (int n = 0; n < total->cnt_line; n++) {
            LINE& line = total->line[n];
            int stops = line.COUNT();
            for (int m = 1; m <stops; m++) {
                POINT s = total->point[line[m - 1]];
                POINT t = total->point[line[m]];
                QLineF ql = QLineF(s.getx(), s.gety(), t.getx(), t.gety());
                scene->addLine(ql, qpenl);
            }
        }
        parnt->setScene(scene);
        parnt->show();
}
```

```
MAP::MAP(QWidget *parent):QMainWindow(parent)
{
    ui.setupUi(this);
    fl = Q_NULLPTR;
    fOrg = Q_NULLPTR;
    BE = Q_NULLPTR;
    gItem= Q_NULLPTR;
    m_Timer = new QTimer(this);
    m_Timer->setSingleShot(true);                    //定时器只执行一次
    connect(ui.action_open, SIGNAL(triggered(bool)), this, SLOT(loadmap()));
    connect(ui.action_exit, SIGNAL(triggered(bool)), this, SLOT(closewnd()));
    connect(ui.action_mintrans, SIGNAL(triggered(bool)), this, SLOT(mintrans()));
    connect(ui.action_cor, SIGNAL(triggered(bool)), this, SLOT(loadOrg()));
    connect(ui.action_begin_end, SIGNAL(triggered(bool)), this, SLOT(beginEnd()));
    //以下 Lambda 表达式可以用自动类型推导代替,或用非成员函数的地址代替
    connect(m_Timer, &QTimer::timeout, this, [=]() {
        QList<QGraphicsItem*> listItem = ui.graphicsView->scene()->items();
        deleteItems();          //查询结果显示时间一到，就删除场景的所有图元
        route = false;          //查询结果显示完毕，可以重新选取步行起点或终点
    });
}
MAP::~MAP(){
    if (fl != Q_NULLPTR) {
        fl->hide();
        delete fl;
        fl = Q_NULLPTR;
        delete m_Timer;
        deleteItems();
        delete total;
    }
}
void MAP::closewnd() {
    if (ui.action_exit->isChecked()==false) return; //鼠标点击一次触发两次，第二次触发直接返回
    ui.action_exit->setChecked(false);   //鼠标第一次触发设置状态为 false,防止第 2 次触发进入
    if (fl != Q_NULLPTR) {
        fl->hide();
        delete fl;
        fl = Q_NULLPTR;
    }
    close();
}
void MAP::closeEvent()
{
    if (fl != Q_NULLPTR) {
        fl->hide();
```

```
            delete fl;
            fl = Q_NULLPTR;
        }
    }
    void MAP::loadmap() {

        if (ui.action_open->isChecked() == false) return; //鼠标点击触发两次，第二次触发直接返回
        ui.action_open->setChecked(false);  //鼠标第一次触发设置状态为 false,防止第 2 次触发进入
        if (fl != Q_NULLPTR) {   //如果先前打开过站点及线路输入窗口
            fl->show();              //则直接显示该窗口
            return;
        }
        arrive=depart = false;   //此时未选取步行起点或终点
        fl = new QtWidgetsFL(); //如果以前没有打开过站点及线路输入窗口
        fl->setWindowFlags(Qt::WindowStaysOnTopHint); //设置最外层显示
        fl->myShow(ui.graphicsView);
    }
    void MAP::loadOrg()
    {
        if (ui.action_cor->isChecked() == false) return; //鼠标点击触发两次，第二次触发直接返回
        ui.action_cor->setChecked(false);   //鼠标第一次触发设置状态为 false,防止第 2 次触发进入
        if (fOrg != Q_NULLPTR) {   //如果先前打开机构输入窗口
            fOrg->show();              //则直接显示该窗口
            return;
        }
        fOrg = new QtWidgetsOrg(); //如果以前没有打开过机构输入窗口
        fOrg->setWindowFlags(Qt::WindowStaysOnTopHint); //设置最外层显示
        fOrg->myShow(ui.graphicsView);
    }
    void MAP::beginEnd()
    {
        if (ui.action_begin_end->isChecked() == false) return; //鼠标点击触发两次，第二次触发直接返回
        ui.action_begin_end->setChecked(false);   //鼠标第一次触发设置状态为 false,防止第 2 次触发进入
        if (BE != Q_NULLPTR) {   //如果先前打开过站点及线路输入窗口
            BE->show();              //则直接显示该窗口
            return;
        }
        arrive = depart = false;   //此时未选取步行起点或终点
        BE = new QtWidgetsBE(); //如果以前没有打开过站点及线路输入窗口
        BE->parnt = ui.graphicsView;
        BE->setWindowFlags(Qt::WindowStaysOnTopHint); //设置最外层显示
        BE->myShow(ui.graphicsView);

    }
    void MAP::deleteItems() { //删除场景的所有图元
```

```
    if (gItem == Q_NULLPTR) return;
    ui.graphicsView->scene()->removeItem(gItem);
    for (int i = gItem->childItems().size() - 1; i >= 0; i--) {
        QGraphicsItem* item = (gItem->childItems())[i];
        gItem->removeFromGroup(item);
        delete item;
    }
    delete gItem;
    gItem = Q_NULLPTR;
}
void MAP::mintrans() {
    //先计算最少转乘的路径,先获得起点坐标和终点坐标
    if (ui.action_mintrans->isChecked() == false) return; //鼠标点击一次触发两次，第二次触发直接返回
    ui.action_mintrans->setChecked(false);  //鼠标第一次触发设置状态为 false,防止第 2 次触发进入
    QList<QGraphicsItem*> listItem;
    if ((depart&&arrive)==false) return;//若没有选中步行起点和终点，则返回
    listItem = ui.graphicsView->scene()->items();
    MyItem* itemDepart = (MyItem*)listItem[0];
    MyItem* itemArrive = (MyItem*)listItem[1];
    if (itemDepart->f() != 1) {              //若不是步行起点，则交换
        itemDepart = (MyItem*)listItem[1];
        itemArrive = (MyItem*)listItem[0];
    }
    //开始组建图元组：用于显示转乘方案的路径
    QGraphicsEllipseItem* myEItem;
    QGraphicsLineItem* myLItem;
    MyScene* scene = (MyScene*)(ui.graphicsView->scene());
    QBrush qbrush(QColor(00, 00, 255));//设置颜色
    QPen qpens(qbrush, 4, Qt::SolidLine, Qt::SBEareCap, Qt::BevelJoin); //站点的笔
    QPen qpenl(qbrush, 3, Qt::SolidLine, Qt::SBEareCap, Qt::BevelJoin); //线路的笔
    route = true;                    //进入查询路径显示期间，不响应步行起点与终点选取
    int c,n = 0;                     //c 为可行转乘方案数，n 为最少转乘次数
    OPTION r[100];                       //一次查询，最多返回 100 条可行转乘方案
    for (int d = 0; d < itemDepart->c(); d++) {   //接近起点的所有公交站点
        int s=(*itemDepart)[d]; //获得起点站点编号 s
        for (int a = 0; a < itemArrive->c(); a++) { //接近终点的所有公交站点
            int t = (*itemArrive)[a];                    //获得终点站点编号 t
            if (s == t) {                                  //起点站和终点站相同不用转乘
                deleteItems();                              //删除先前的转乘方案路径显示
                gItem = new QGraphicsItemGroup();
                myEItem = new QGraphicsEllipseItem(itemDepart->x(), itemDepart->y(), 6, 6);
                myEItem->setPen(qpens);
                myEItem->setBrush(qbrush);
                gItem->addToGroup(myEItem);              //步行起点的位置
                myLItem = new QGraphicsLineItem(itemDepart->x(), itemDepart->y(), itemArrive->x(), itemArrive->y());
```

```
                myLItem->setPen(qpenl);
                gItem->addToGroup(myLItem);              //到步行终点的路径
                myEItem = new QGraphicsEllipseItem(itemArrive->x(), itemArrive->y(), 6, 6);
                myEItem->setPen(qpens);
                myEItem->setBrush(QBrush(QColor(255, 170, 00)));
                gItem->addToGroup(myEItem);              //步行终点的位置
                scene->addItem(gItem);
                ui.graphicsView->setScene(scene);
                continue;
            }
            c=TOTAL::tra.miniTran(s, t, n, r);           //得到的转乘方案数
            for (int m = 0; m < c; m++){                 //对于第 m 个转乘方案即 route=r[m], 转乘次数都为 n
                deleteItems();
                gItem = new QGraphicsItemGroup();
                myEItem = new QGraphicsEllipseItem(itemDepart->x(), itemDepart->y(), 6, 6);
                myEItem->setPen(qpens);
                myEItem->setBrush(qbrush);
                gItem->addToGroup(myEItem);              //步行起点的位置
                int fr = s, to = t;                      //起始站 fr 与终点站 to
                int fm, tm;                              //已乘线路 fm 与 z 转乘线路 tm
                int bg, ed;                              //线路中的起始站点序号 bg,终止站点序号 ed
                myLItem = new QGraphicsLineItem(itemDepart->x(), itemDepart->y(), total->point[s].getx(),
total->point[s].gety());
                myLItem->setPen(qpenl);
                gItem->addToGroup(myLItem);              //到起点站
                if (n == 1 && r[m][0].getpoint() == -1) {  //即从 i 路到 i 路(此时 S()==-1)不用转乘
                    fm = r[m][0].getbegin();             //已乘线路序号 fm
                    bg = TOTAL::line[fm].has(fr);        //起始站点在线路中的序号
                    ed = TOTAL::line[fm].has(to);        //终止站点在线路中的序号
                    if (bg > ed) { tm = bg; bg = ed; ed = tm; }
                    for (int y = bg; y < ed; y++)    //从起始站点下一序号到终止站点序号
                    {
                        fr= TOTAL::line[fm][y];          //得到该站点序号对应的站点编号
                        to= TOTAL::line[fm][y + 1];      //得到该站点序号对应的站点编号
                        myLItem = new QGraphicsLineItem(TOTAL::point[fr].getx(), TOTAL::point[fr].gety(),
TOTAL::point[to].getx(), TOTAL::point[to].gety());
                        myLItem->setPen(qpenl);
                        gItem->addToGroup(myLItem); //到下一站的路径
                    }
                }
                else {
                    for (int y = 0; y < n; y++)      //对于每个转乘
                    {
                        fm = r[m][y].getbegin();             //对于每个转乘的起始线路
                        bg = TOTAL::line[fm].has(fr);
```

```
                    to = r[m][y].getpoint();                    //对于起始线路的转乘站点
                    ed = TOTAL::line[fm].has(to);
                    if (bg > ed) { tm = bg; bg = ed; ed = tm; }
                    for (int u = bg; u < ed; u++)   //从起始站点下一序号到终止站点序号
                    {
                        int ff = TOTAL::line[fm][u];      //得到该站点序号对应的站点编号
                        int tt = TOTAL::line[fm][u+1];   //得到该站点序号对应的站点编号
                        myLItem = new QGraphicsLineItem(TOTAL::point[ff].getx(), TOTAL::point[ff].gety(),
TOTAL::point[tt].getx(), TOTAL::point[tt].gety());
                        myLItem->setPen(qpenl);
                        gItem->addToGroup(myLItem); //到下一站的路径
                    }
                    fr = to;                                    //作为下一起点
                }
                fm = r[m][n-1].getend();                        //对于最后乘坐的线路
                bg = TOTAL::line[fm].has(fr);
                ed = TOTAL::line[fm].has(t);
                if (bg > ed) { tm = bg; bg = ed; ed = tm; }
                for (int u = bg; u < ed; u++)           //从起始站点下一序号到终止站点序号
                {
                    int ff = TOTAL::line[fm][u];            //得到该站点序号对应的站点编号
                    int tt = TOTAL::line[fm][u + 1];      //得到该站点序号对应的站点编号
                    myLItem = new QGraphicsLineItem(TOTAL::point[ff].getx(), TOTAL::point[ff].gety(),
TOTAL::point[tt].getx(), TOTAL::point[tt].gety());
                    myLItem->setPen(qpenl);
                    gItem->addToGroup(myLItem);        //到下一站的路径
                }
            }
            myLItem = new QGraphicsLineItem(TOTAL::point[t].getx(), TOTAL::point[t].gety(), itemArrive->x(),
itemArrive->y());
            myLItem->setPen(qpenl);
            gItem->addToGroup(myLItem);                    //到步行终点的路径
            myEItem = new QGraphicsEllipseItem(itemArrive->x(), itemArrive->y(), 6, 6);
            myEItem->setPen(qpens);
            myEItem->setBrush(QBrush(QColor(00, 00, 255)));
            gItem->addToGroup(myEItem);                    //步行终点的位置
            scene->addItem(gItem);
            ui.graphicsView->setScene(scene);
        }
    }
}
    this->m_Timer->start(5000);   //展示查询的路径结果 5 秒
}
```

QtTipsDlgView.cpp:

```
#include "QtTipsDlgView.h"

QtTipsDlgView::QtTipsDlgView(const QString& msg, QWidget *parent)
	: QDialog(parent)
{
	ui.setupUi(this);
	setWindowFlags(Qt::FramelessWindowHint | Qt::Tool | Qt::WindowStaysOnTopHint);
	setAttribute(Qt::WA_TranslucentBackground);

	initFrame(msg);

	m_pTimer = new QTimer(this);
	m_pTimer->setSingleShot(true);    //定时器只执行一次
	connect(m_pTimer, &QTimer::timeout, this, [=]() {this->close(); });
}

QtTipsDlgView::~QtTipsDlgView()
{
	if (this->m_pTimer != Q_NULLPTR)
	{
		this->m_pTimer->deleteLater();
	}
}

void QtTipsDlgView::startTimer(int ms)
{
	this->m_pTimer->start(ms);
}
void QtTipsDlgView::initFrame(const QString& msg)
{
	ui.labelMsg->setText(msg);
}

QtWidgetsFL.cpp:
#include <QFileDialog>
#include <QMessageBox>
#include "QtWidgetsFL.h"
#include "fun.h"
#include "map.h"
//自定义的站点及线路输入界面
QtWidgetsFL::QtWidgetsFL(QWidget *parent): QWidget(parent)
{
	ui.setupUi(this);
	connect(ui.pushButtonStop, SIGNAL(clicked()), this, SLOT(inputStop()), Qt::UniBEeConnection);
	connect(ui.pushButtonLine, SIGNAL(clicked()), this, SLOT(inputLine()), Qt::UniBEeConnection);
```

```
    connect(ui.pushButtonDone, SIGNAL(clicked()), this, SLOT(checkFile()), Qt::UniBEeConnection);
    connect(ui.pushButtonBEit, SIGNAL(clicked()), this, SLOT(close()), Qt::UniBEeConnection);
}
void QtWidgetsFL::myShow(QGraphicsView* p) {
    parnt = p;
    show();
}
QtWidgetsFL::~QtWidgetsFL()
{
}
void QtWidgetsFL::inputStop() {
    ui.labelHits->setText("");
    QString fileName = QFileDialog::getOpenFileName(this, tr("Open File"), ".", tr("*.txt"));
    ui.textEditStop->setText(fileName);
}
void QtWidgetsFL::inputLine() {
    ui.labelHits->setText("");
    QString fileName = QFileDialog::getOpenFileName(this, tr("Open File"), "lines", tr("*.txt"));
    ui.textEditLine->setText(fileName);
}

void QtWidgetsFL::checkFile() {
    QString    fs = ui.textEditStop->toPlainText();
    QString    fl = ui.textEditLine->toPlainText();
    if (fs.isEmpty() && fl.isEmpty()) {
        ui.labelHits->setStyleSheet("color: rgb(255, 0, 0)");
        ui.labelHits->setText(QString::fromOrgal8Bit("操作提示：没有输入站点及线路文件路径！"));
        ui.textEditStop->setFocus();
        return;
    }
    if (fs.isEmpty()) {
        ui.labelHits->setStyleSheet("color: rgb(255, 0, 0)");
        ui.labelHits->setText(QString::fromOrgal8Bit("操作提示：没有输入站点文件路径！"));
        ui.textEditStop->setFocus();
        return;
    }
    if (fl.isEmpty()) {
        ui.labelHits->setStyleSheet("color: rgb(255, 0, 0)");
        ui.labelHits->setText(QString::fromOrgal8Bit("操作提示：没有输入线路文件路径！"));
        ui.textEditLine->setFocus();
        return;
    }
    //处理站点文件
    ui.labelHits->setStyleSheet("color: rgb(255, 0, 0)");  //设置操作提示信息显示颜色
    ui.labelHits->setText(QString::fromOrgal8Bit("操作提示：正在处理站点和线路文件..."));
```

```
        try {//读入站点及线路文件并初始化
            if (total != nullptr) delete total;
            total = new TOTAL(fs.toStdString().c_str(), fl.toStdString().c_str());
            ((MyScene*)(parnt->scene()))->stopLines(parnt); //在背景地图上画出站点及公交线路
        }
        catch (...) {//读入或初始化失败
            total = nullptr;
            close();
            QMessageBox::information(NULL, QString::fromOrgal8Bit("操作提示"), QString::fromOrgal8Bit("公交站点或公交线路文件读入及初始化失败！"));
        }
        ui.labelHits->setText("");
        close();
    }
```

QtWidgetsOrg.cpp:

```
#include "QtWidgetsOrg.h"
#include <QFileDialog>
#include <QMessageBox>
#include "fun.h"
#include <iostream>
#include <map>
#include <string>
#include <fstream>
#include "map.h"
using namespace std;
map<string, pair<int, int> > Org;
QtWidgetsOrg::QtWidgetsOrg(QWidget *parent)
    : QWidget(parent)
{
    ui.setupUi(this);
    connect(ui.pushButtonOrg, SIGNAL(clicked()), this, SLOT(inputOrg()), Qt::UniBEeConnection);
    connect(ui.pushButtonDone, SIGNAL(clicked()), this, SLOT(checkFile), Qt::UniBEeConnection);
    connect(ui.pushButtonBEit, SIGNAL(clicked()), this, SLOT(close()), Qt::UniBEeConnection);
}

void QtWidgetsOrg::myShow(QGraphicsView* p)
{
    parnt = p;
    show();
}

QtWidgetsOrg::~QtWidgetsOrg()
{
}
```

```
void QtWidgetsOrg::inputOrg() {
	ui.labelHits->setText("");
	QString fileName = QFileDialog::getOpenFileName(this, tr("Open File"), ".", tr("*.txt"));
	ui.textEditOrg->setText(fileName);
}

void QtWidgetsOrg::checkFile() {
	QString	fs = ui.textEditOrg->toPlainText();
	if (fs.isEmpty())
	{
		//如果字符串为空
		ui.labelHits->setStyleSheet("color: rgb(255, 0, 0");
		ui.labelHits->setText(QString::fromOrgal8Bit("操作提示：没有输入地点文件路径！"));
		ui.textEditOrg->setFocus();
		return;
	}
	//处理地点文件
	ui.labelHits->setStyleSheet("color: rgb(255, 0, 0)");  //设置操作提示信息显示颜色
	ui.labelHits->setText(QString::fromOrgal8Bit("操作提示：正在处理地点文件..."));
	loadOrgFile(fs.toStdString());
	ui.labelHits->setText("");
	close();
}



void loadOrgFile(string fs)
{
	string filename = fs;
	string str;
	int x, y;
	char ch;

	ifstream infile;
	infile.open(filename);

	while (infile >> str)
	{
		infile >> x >> ch >> y;
		Org.insert(pair<string, pair<int, int> >(str, pair<int, int>(x, y)));
	}
	infile.close();

}
```

QtWidgetsBE.cpp:

```
#include "QtWidgetsOrg.h"
#include <QFileDialog>
#include <QMessageBox>
#include "fun.h"
#include <iostream>
#include <map>
#include <string>
#include <fstream>
#include "map.h"
using namespace std;
map<string, pair<int, int> > Org;
QtWidgetsOrg::QtWidgetsOrg(QWidget *parent)
	: QWidget(parent)
{
	ui.setupUi(this);
	connect(ui.pushButtonOrg, SIGNAL(clicked()), this, SLOT(inputOrg()), Qt::UniBEeConnection);
	connect(ui.pushButtonDone, SIGNAL(clicked()), this, SLOT(checkFile), Qt::UniBEeConnection);
	connect(ui.pushButtonBEit, SIGNAL(clicked()), this, SLOT(close()), Qt::UniBEeConnection);
}

void QtWidgetsOrg::myShow(QGraphicsView* p)
{
	parnt = p;
	show();
}

QtWidgetsOrg::~QtWidgetsOrg()
{
}

void QtWidgetsOrg::inputOrg() {
	ui.labelHits->setText("");
	QString fileName = QFileDialog::getOpenFileName(this, tr("Open File"), ".", tr("*.txt"));
	ui.textEditOrg->setText(fileName);
}

void QtWidgetsOrg::checkFile() {
	QString	fs = ui.textEditOrg->toPlainText();
	if (fs.isEmpty())
	{
		//如果字符串为空
		ui.labelHits->setStyleSheet("color: rgb(255, 0, 0");
		ui.labelHits->setText(QString::fromOrgal8Bit("操作提示：没有输入地点文件路径！"));
```

```
            ui.textEditOrg->setFocus();
            return;
        }
        //处理地点文件
        ui.labelHits->setStyleSheet("color: rgb(255, 0, 0)");  //设置操作提示信息显示颜色
        ui.labelHits->setText(QString::fromOrgal8Bit("操作提示：正在处理地点文件..."));
        loadOrgFile(fs.toStdString());
        ui.labelHits->setText("");
        close();
}



void loadOrgFile(string fs)
{
        string filename = fs;
        string str;
        int x, y;
        char ch;

        ifstream infile;
        infile.open(filename);

        while (infile >> str)
        {
            infile >> x >> ch >> y;
            Org.insert(pair<string, pair<int, int> >(str, pair<int, int>(x, y)));
        }
        infile.close();

}
```