



哈尔滨工业大学(威海)
Harbin Institute of Technology, Weihai

软件设计与开发实践 II

课程报告

任务题目： 哈尔滨工业大学（威海）
校园导航系统

学 号： 180400715

姓 名： 杨卓宸

组 号： 1804102

任课教师： 张华

哈尔滨工业大学（威海）计算机科学与技术学院

前 言

《软件设计与开发实践 II》是基于自选项目的实践训练，学生将综合利用《集合论与图论》、《数据结构》、《算法设计与分析》、《高级语言程序设计 I 及 II》、《Java 程序设计》等方面的基本概念、原理、技术和方法，开展实际应用问题设计求解和对应系统软件开发两大方面的实践。

通过本课程的学习、训练和实践，引导学生熟练掌握问题设计求解和软件编程开发的一般过程、相关技术、方法和途径；训练综合运用所学的理论知识和方法独立分析和解决问题，提高问题分析、问题求解和软件开发能力；培养学生能够针对实际问题，选择适当的数据结构、设计有效算法，提高程序设计的能力和编码质量；训练和学会用系统的观点和软件开发一般规范进行软件设计开发，培养软件工作者所应具备的科学工作方法和作风，提高工程素质；并通过采用团队协作、构建项目组的形式，来培养学生的团队合作与交流能力。

本课程要求学生分组进行（每组 1~2 人），通过一定的调研来自行结合实际应用需求来选题，并由任课教师来对学生选题做筛选评定。要求所设计开发的软件具有一定的实用性和系统完整性，要有较友好的图形交互操作界面，并对输入数据有较强的完整性约束，要以用户需求作为出发点来设计软件界面和功能模块。本课程主要教学环节包括：学生自选任务、开题检查、中期检查、软件验收、任务报告撰写提交和任务资料整理归集等。

报告评价 等级	A+	A	B+	B	C+	C
	D+	D	E+	E	F+	F

1. 选题背景与意义

今年，2020 年，哈尔滨工业大学已经建校百年，是一所真正意义上的百年名校了。我们哈尔滨工业大学威海校区也是日新月异，飞速发展。最近学校在现代化建设上也下了很大功夫。在校期间，我发现，校内的会议和讲座变得越来越多。针对学校现代化的实现，为了来访我校的访客能够更方便的了解学校的景点，更顺利的找到校内地点，并且为了方便看校的学生和家长参观，同时减少导游人员的数量，我编写了这个哈尔滨工业大学（威海）校园导航系统。

现如今，算法设计与分析和数据结构，这两者的合二为一对于程序的实现起着非常重要的作用，算法是程序的核心，数据结构是程序的基石，我们完全可以乘科技发展的东风，为智能化的新生活而奋斗，努力实现我们理想的社会生活。而数据结构和算法相关知识的学习，给予了我这个条件，更好地服务方便了人们在较大校园面积的找地儿难问题。通过编写这个程序，我加深了数据结构和算法的了解，巩固了 C++ 和 QT Creator 的知识，同时也为我校加快智能化进程贡献了一份力。

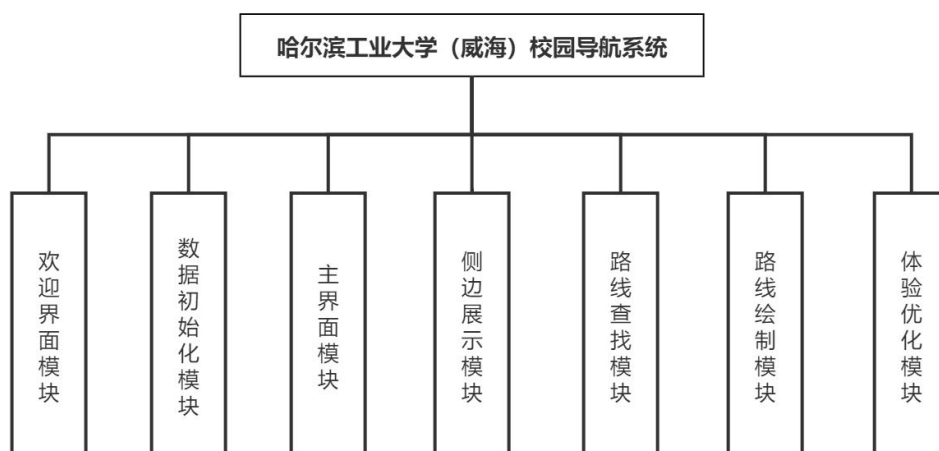
需求分析

所谓校园导航系统，应与我们日常手机电脑上能够接触到的地图导航 APP 很相似。在我的设计思路里，哈尔滨工业大学（威海）校园导航系统是在 Windows 平台通用的应用程序，其应包含可视化的 UI 视图，部分导航按键和显示区域。我认为，这是一个针对我们哈尔滨工业大学威海校区的，功能更丰富的地图导航 APP。所以首先，校园导航系统需要拥有地图导航 APP 的基本功能：地图的 UI 显示，路线规划，路线查找的 UI 重点标记，关键地点的 UI 显示；其次，校园导航系统需要拥有更丰富的介绍功能：关键地点的介绍、图片显示，威海校区的历史介绍，地点的具体信息情况；最终，为了作用最大化，校园导航系统应该易于上手操作，所以我认为最重要的功能有：按键捕捉，突出功能显示，突出路线信息显示等。

此校园导航系统的完整性是指数据的精确和可靠性。其包括实体完整性：规定图的每一节点在图中是唯一的实体。域完整性：是指图中的节点必须满足特定的数据类型约束，规定用邻接矩阵和顶点集合进行存储。参照完整性：是指路线点集和无向图的主关键字和外关键字的数据应一致，保证了数据的一致性和一一对应。用户定义的完整性：能够完成路线查找并显示的基本功能。

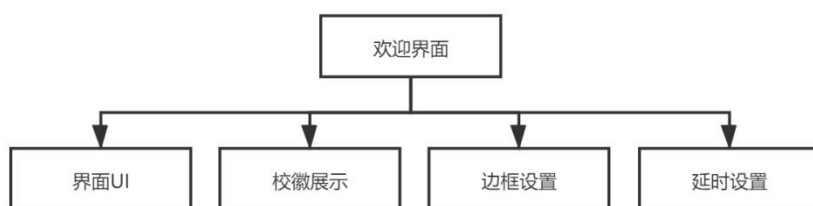
系统主要功能设计

根据校园导航系统的需求，我将其分为几个主要模块：



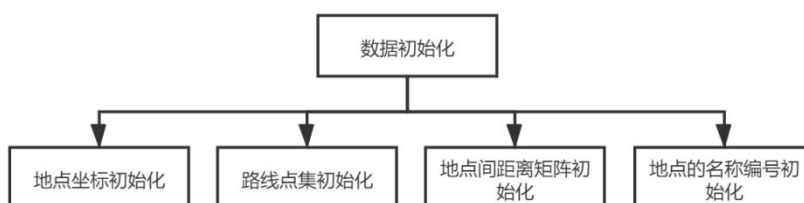
1. 欢迎界面模块

包含界面的 UI 设置，校徽图片展示，边框窗口的透明化和存在时间的设定。



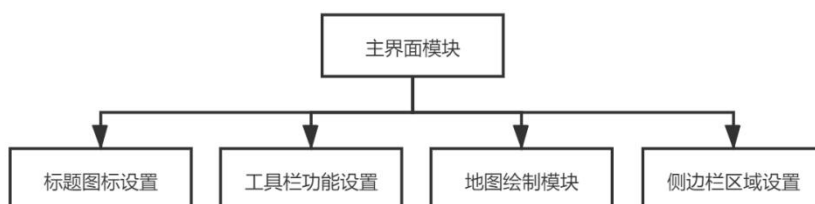
2. 数据初始化模块

包含地点坐标，地点间距离，路线点集，无向图，地点信息的初始化 `createGraph()`。



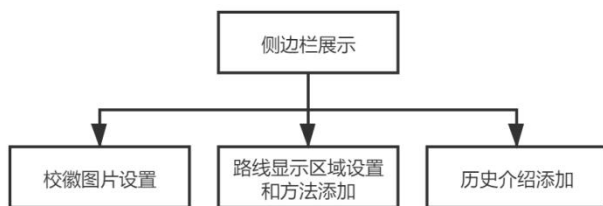
3. 主界面模块

包含标题，图标，工具栏 `createToolBar()`，动作方法 `createAction()`，地图的设定和侧边栏区域的留白。



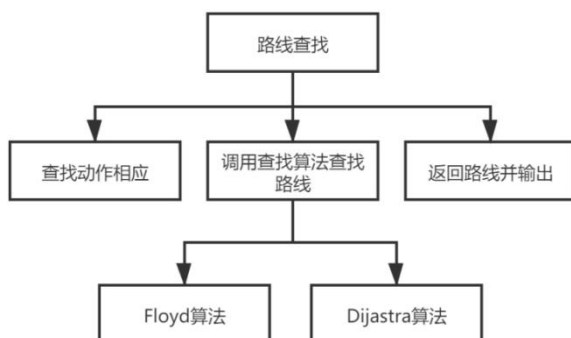
4. 侧边展示模块

侧边展示包含在 UI 文件内添加校徽图片和历史介绍，利用 QLabel 显示路线信息。



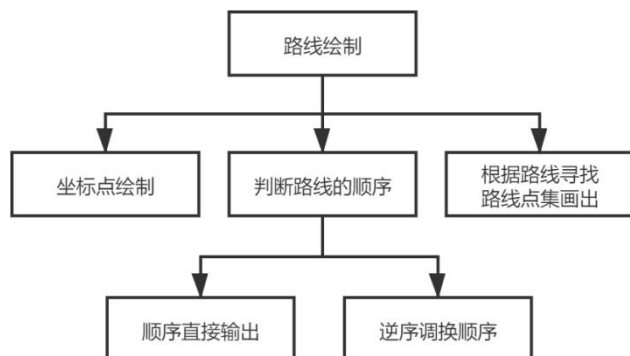
5. 路线查找模块

按下按钮触发动作相应 FindPath(), 调用查找算法并返回路线点 ShortestPath()。



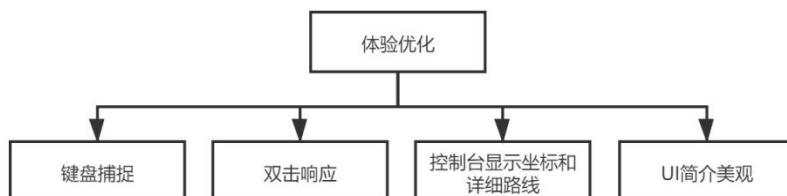
6. 路线绘制模块

DrawRoad()利用 QPainter 判断路线顺序画出精确路线，paintEvent()画出坐标点。



7. 体验优化模块

mousePressEvent(), mouseDoubleClickEvent(), keyPressEvent(), 图片的缩放一致性，用于实现细节上的功能，完善用户体验。



核心算法设计与分析

主要数据结构

1. 无向图

校园导航模型由地点位置和地点之间的路径组成，采用数据结构中的无向图来模拟存储。图的顶点代表校内重要地点，图的边代表景点之间的路径，邻接矩阵存储边的权值，边权代表景点间的距离。

2. 类的邻接矩阵（路线点集）

路线的具体实现是存储路线的坐标位置，每条路线都包含了 5-10 个坐标参考点，用于表示存储路线。路线点集使用类的邻接矩阵实现，方便将每条路线的路线点集存储在一起方便查找。

3. 队列（Vectory 容器）

图中的地点有 ID 和 Name 两个属性，地点的 ID 代表地点次序，Name 代表景点名称，路线的具体坐标点使用队列进行存储。

队列用 Vectory 容器进行实现。Vectory 容器是由 vector 继承并重载而得，实现了控制台输出的功能。

核心算法

1. 查找路径长度和最短路线时，用 Floyd 算法和 Dijkstra 算法实现。

Floyd

时间复杂度： $O(N^3)$ 。

空间复杂度： $O(N^2)$ 。

伪代码：

```
void ShortestPath_FLOYD(MGraph G, PathMatrix &P[], DistancMatrix &D){
    // 用 Floyd 算法求有向网 G 中各对顶点 v 和 w 之间的最短路径 P[v][w] 及其
    // 带权长度 D[v][w]。若 P[v][w][u] 为 TRUE，则 u 是从 v 到 w 当前求得最短路径上的顶点
    for(v = 0; v < G.vexnum; ++ v) // 各对结点之间初始已知路径及距离
        for(w = 0; w < G.vexnum; ++ w){
            D[v][w] = G.arcs[v][w];
            for(u = 0; u < G.vexnum; ++ u) P[v][w][u] = FALSE;
            if(D[v][w] < INFINITY){ // 从 v 到 w 有直接路径
                P[v][w][v] = TRUE; P[v][w][w] = TRUE;
            } // if
        } // for
    for(u = 0; u < G.vexnum; ++ u)
        for(v = 0; v < G.vexnum; ++ v)
            for(w = 0; w < G.vexnum; ++ w)
                if(D[v][u] + D[u][w] < D[v][w]){ // 从 v 经 u 到 w 的一条路径更短
                    D[v][w] = D[v][u] + D[u][w];
                    for(i = 0; i < G.vexnum; ++ i)
                        P[v][w][i] = P[v][u][i] || P[u][w][i];
                } // if
} // ShortestPath_FLOYD
```

Dijkstra

时间复杂度： $O(N)+O(1)+O(N)+O(N^2)=O(N^2)$ 。

空间复杂度： $O(E)$ 。

伪代码：

```
void ShortestPath_DIJ( MGraph G, int v0, PathMatrix &P, ShortPathTable &D){
    // 用 Dijkstra 算法求有向网 G 的 v0 顶点到其余顶点 v 的最短路径 P[v] 及其带权长度 D[v]。
    // 若 P[v][w] 为 TRUE，则 w 是从 v0 到 v 当前求得最短路径上的顶点。
    // final[v] 为 TRUE 当且仅当  $v \in S$ ，即已经求得从 v0 到 v 的最短路径。
    for (v = 0; v < G.vexnum; ++ v){
        final[v] = FALSE; D[v] = G.arcs[v0][v];
        for (w = 0; w < G.vexnum; ++ ww) P[v][w] = FALSE; // 设空路径
        if (D[v] < INFINITY) {P[v][v0] = TRUE; P[v][v] = TRUE;}
    } // for
    D[v0] = 0; final[v0] = TRUE; // 初始化，v0 顶点属于 S 集
    // 开始主循环，每次求得 v0 到某个 v 顶点的最短路径，并加 v 到 S 集
    for (i = 1; i < G.vexnum; ++ i) { // 其余 G.vexnum - 1 个顶点
```



```

    min = INFINITY; // 当前所知离 v0 顶点的最近距离
    for (w = 0; w < G.vexnum; ++ w)
        if(!final[w]) // w 顶点在 V-S 中
            if(D[w] < min) {v = w; min = D[w];} // w 顶点离 v0 顶点更近
    final[v] = TRUE; // 离 v0 顶点最近的 v 加入 S 集
    for(w = 0; w < G.vexnum; ++ w) // 更新当前最短路径及距离
        if(!final[w] && (min + G.arcs[v][w] < D[w])) { // 修改 D[w] 和 P[w], x ∈ V-S
            D[w] = min + G.arcs[v][w];
            P[w] = P[v]; P[w][w] = TRUE; // P[w] = P[v] + P[w]
        } // if
    } // for
} // ShortestPath_DIJ

```

2. 查找路线对应的路线点集时，采用**插值查找**算法。

时间复杂度：查找成功或者失败的时间复杂度均为 $O(\log_2(\log_2 N))$ 。

空间复杂度：没有临时变量参与，空间复杂度为 $O(1)$ 。

核心公式： $mid = low + ((key - A[low]) / (A[high] - A[low])) * (high - low)$

描述：假设表中有 n 个元素，查找过程为取区间中间元素的下标 mid ，对 mid 的关键字与给定值的关键字比较：

- (1) 如果与给定关键字相同，则查找成功，返回在表中的位置；
- (2) 如果给定关键字大，则更新左区间起始位置等于 $mid + 1$ ，即向右查找；
- (3) 如果给定关键字小，查找的值不在范围，直接返回；
- (4) 重复过程，直到找到关键字或失败。

通常情况下：返回值，代表下标；返回-1，代表没有找到关键字；

伪代码：

```

int InsertionSearch( int A[], int high, int key, int low)
//high 当前数组下标的最大值，low 为当前数组下标的最小值；
{
    int mid = low + ((key - A[low]) / (A[high] - A[low])) * (high - low);
    if(key == A[low])
        return mid;
    if(key > A[mid])
        return InsertionSearch( A, high, key, mid+1);
    if(key < A[mid])
        return InsertionSearch( A, mid-1, key, low);
}

```

2. 系统核心模块设计

哈尔滨工业大学（威海）校园导航系统使用 C++ 语言编写，编程支撑软件为 Qt Creator 4.3.1，Based on Qt 5.9.0 (MSVC 2015, 32 bit)，开发环境编译器为 MinGW 32bit，8.00GB RAM，运行系统为 Windows 10 Professional Version 1909 64bit，基于 Intel Core i5-7360U x64 CPU，软件的系统架构采用事件驱动架构，通过鼠标的点击和按钮的点击事件，触发查找动作，处理路线显示功能。

主要数据结构定义代码

1. 无向图

```
class MGraph{
public:
    int vertex[MAX_VERTEX_NUM];           //顶点集合
    int arcs[MAX_VERTEX_NUM][MAX_VERTEX_NUM]; //邻接矩阵
    int verNum;                           //顶点数
    int arcNum;                           //边数
};
```

2. 类的邻接矩阵

```
class MRoad{
public:
    int vertex[MAX_ROAD_NUM][2];         //路线点集合
    int verNum;                           //点数
};
```

3. 队列

```
template<class T>class Vectory:public QVector<T>{
public:
    virtual void push_back(T &t){         //入队
        append(std::move(t));
        cout << t << endl;
    }
    virtual void push_front(const T &t){ //出队
        prepend(t);
        cout << t << endl;
    }
};
```

核心函数定义

```
public:
    void createToolBar();           //生成工具栏
    void createAction();           //生成查找清屏动作

public slots:
    void FindPath();               //查找动作
    void Clear();                 //清屏动作

private:
    void CreateGraph();            //生成无向图
    void InitMap();               //生成整个地图
    void DrawRoad(QPainter& painter); //画出路线
    bool Calculate();              //查找最短路径的调用函数
        //返回存在与否
    static QPoint getPosition(int id); //id 为地点编号
        //得到对应 id 的地点的坐标
        //返回 QPoint 坐标点
    static int getId(int x,int y);    //x, y 为横纵坐标
        //得到对应坐标的地点
        //返回地点编号 id
    static QString getName(int id);    //id 为地点编号
        //得到对应 id 的地点名称
        //返回 QString 地点名称
    int getRoad(int a[], int value, int low, int high);
//a[]为被查找集, value 为查询值, low 和 high 为左右索引
    //查找对应路线的路线点集
    //返回路线编号
    bool ShortestPath(MGraph &G,int start,int end,QVector<int>& R);
//G 为无向图, start 为路线起点, wnd 为路线终点, R 为路线经过地点集
    //查找 start 至 end 的最短路径并输出
    //返回路线的存在与否

protected:
    void mouseDoubleClickEvent (QMouseEvent *e); //双击显示图片介绍
    void paintEvent(QPaintEvent *event);        //画出坐标点和路线
    void mousePressEvent(QMouseEvent *e);        //控制台输出坐标
    void keyPressEvent(QKeyEvent *event);        //键盘监测捕捉
```

核心函数具体代码

1. 插值查找

```
int MainWindow::getRoad(int a[], int value, int low, int high)
{
    int mid = low+(value-a[low])/(a[high]-a[low])*(high-low);
    if(a[mid]==value)
        return mid;
    if(a[mid]>value)
        return getRoad(a, value, low, mid-1);
    if(a[mid]<value)
        return getRoad(a, value, mid+1, high);
}
```

2. 查找最短路径

```
bool MainWindow::ShortestPath(MGraph &G,int start,int end,QVector<int>& R)
{
    if(start == end)return false;
    const int N = G.verNum, n = N;
    int Path[N][N];          //表示 vi 和 vj 之间的最短路上的前驱顶点
    long long D[N][N];       //表示 vi 和 vj 之间的最短路径长度
    int Place[N];
    if(start + end > -3)
    {
        //Floyd
        for(int i = 0; i < N ; ++ i)
        {
            for(int j = 0; j < N; ++ j)
            {
                D[i][j] = G.arcs[i][j];
                Path[i][j] = D[i][j] != INF ? i : -1;
            }
        }

        for(int k = 0; k < N; ++ k)
        {
            for(int i = 0; i < N; ++ i)
            {
                for(int j = 0; j < N; ++ j)
                {
                    if(D[i][k] + D[k][j] < D[i][j])
                    {
                        D[i][j] = D[i][k] + D[k][j];
                        Path[i][j] = Path[k][j];
                    }
                }
            }
        }
    }
}
```

```

    }
}
else
{
//Dijkstra
    for (int k = 0; k < n; k++)
    { //控制总循环的次数
        int Min = INF;
        int pos = 1; //min 记录最小路径, pos 记录下一个访问结点的标
        //找到最短距离, 以及对应下标, 作为下一次的起始节点
        for (int i = 0, j = 0; i < n; i++)
        {
            if ((Place[i] == 0) && (Place[j] < Min))
            {
                Min = Place[i];
            }
        }
        if (pos == 1) break;
        //根据下一个结点在矩阵中对应的值, 调整结点到原点距离的最小值
        for (int j = 0; j < n; j++)
        {
            if ((Place[j] == 0) && (D[pos][j] != INF)) //判断 D 中对应的值是否存在
            {
                if (Place[j] > Place[pos] + D[pos][j])
                {
                    Place[j] = Place[pos];
                }
                /*else if (city[j].dist == (city[pos].dist + map[pos][j]))
                //这种情况是为了处理最短路径有多条时的情况
                {
                    city[j].number += city[pos].number;
                    if (city[j].call < city[pos].call + callnum[j])
                    city[j].call = city[pos].call + callnum[j];
                }*/
            }
        }
    }
}

/* 输出每对最短路径 */
for (int i = 0; i < N; ++i)
{
    for (int j = 0; j < N; ++j)
    {
        if (G.vertex[i] == start && G.vertex[j] == end)

```

```

    {
        R.clear();
        R.push_front(end);
        //显示
        cout << G.vertex[i] << " to " << G.vertex[j] << ":" << G.vertex[j];
        for(int vi = Path[i][j]; vi != i; vi = Path[i][vi])
        {
            cout << " to " << G.vertex[vi];
            R.push_front(G.vertex[vi]);
        }
        cout << " to " << G.vertex[i] << " D[i][j]*2+116" << endl;
        d = D[i][j];
        R.push_front(start);
        return R.size()>=2;
    }
}
return R.size()>=2;
}

```

3. 路线生成显示

```
void MainWindow::DrawRoad(QPainter &painter)
```

```

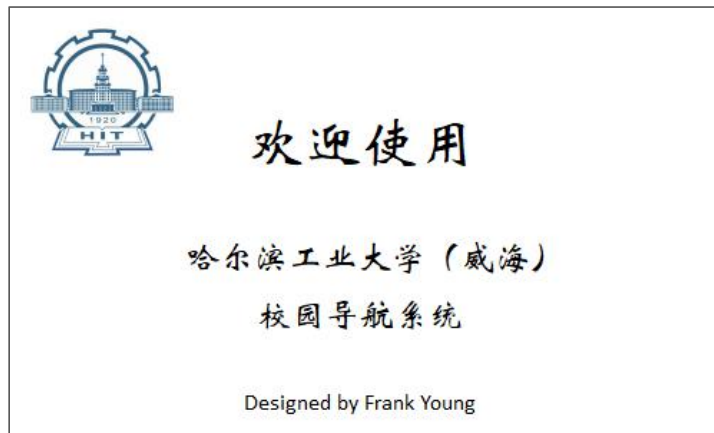
{
    painter.setPen(QPen(Qt::red,5,Qt::DotLine,Qt::RoundCap,Qt::MiterJoin));
    //painter.setPen("black");
    for(int i=1;i<road.size();i++)
    {
        if(road[i-1]>road[i])
        {
            painter.drawLine(getPosition(road[i-1]),QPoint(r[road[i]][road[i-1]].vertex[r[road[i]][road[i-1]].verNum-1][0],r[road[i]][road[i-1]].vertex[r[road[i]][road[i-1]].verNum-1][1]));
            for(int j=r[road[i]][road[i-1]].verNum-1;j>0;j--)
            {
                painter.drawLine(QPoint(r[road[i]][road[i-1]].vertex[j][0],r[road[i]][road[i-1]].vertex[j][1]),QPoint(r[road[i]][road[i-1]].vertex[j-1][0],r[road[i]][road[i-1]].vertex[j-1][1]));
            }
            painter.drawLine(QPoint(r[road[i]][road[i-1]].vertex[0][0],r[road[i]][road[i-1]].vertex[0][1]),getPosition(road[i]));
        }
        else
        {
            painter.drawLine(getPosition(road[i-1]),QPoint(r[road[i-1]][road[i]].vertex[0][0],r[road[i-1]][road[i]].vertex[0][1]));
            for(int j=0;j<r[road[i-1]][road[i]].verNum-1;j++)

```

```
{  
    painter.drawLine(QPoint(r[road[i-1]][road[i]].vertex[j][0],r[road[i-1]][road[i]].vertex[j][1]),QPoint(r[road[i-1]][road[i]].vertex[j+1][0],r[road[i-1]][road[i]].vertex[j+1][1]));  
}  
    painter.drawLine(QPoint(r[road[i-1]][road[i]].vertex[r[road[i-1]][road[i]].verNum-1][0],r[road[i-1]][road[i]].vertex[r[road[i-1]][road[i]].verNum-1][1]),getPosition(road[i]));  
}  
}
```

主要功能截图

1. 欢迎窗口



开始运行程序，首先生成展示欢迎窗口。

欢迎窗口意在欢迎用户的使用，营造用户的高级体验感。

欢迎窗口会持续存在 3s，自动消失。

2. 主界面窗口



主界面窗口在欢迎窗口消失后自动打开，主要分为三个部分。

- (1) 地图显示部分：将地图绘制在显示区域，并将地点用蓝点绘制，突出坐标点位置，并初始化各个数据点。
- (2) 工具栏部分：添加起点的下拉选项菜单和终点的下拉选项菜单，添加导航动作按钮和清屏重置动作按钮。
- (3) 右边栏部分：添加校徽的图片显示，导航系统的名称，哈工大威海的学校简介，起点终点距离的数据显示区域。

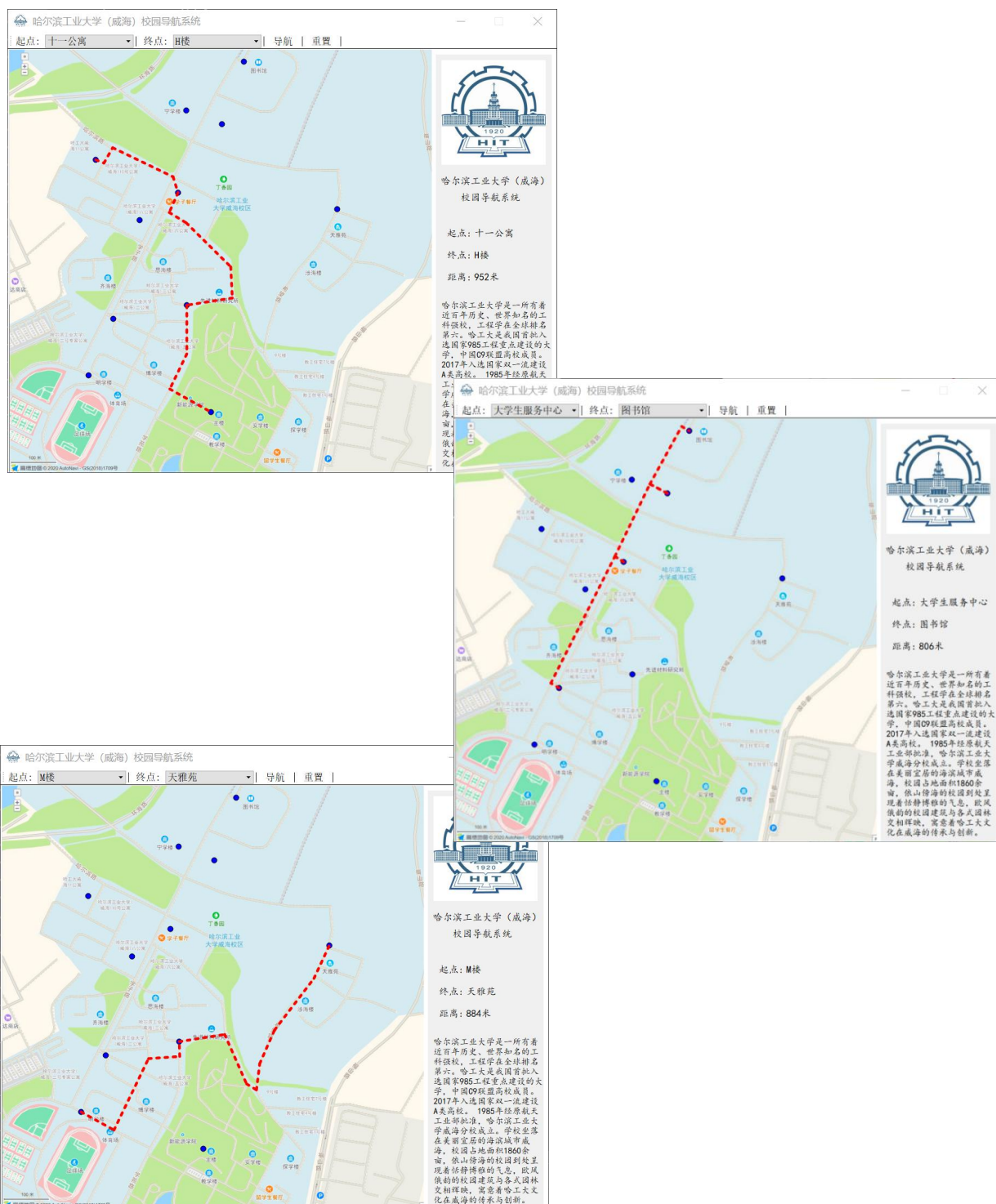
在主界面窗口，我将 11 公寓和图书馆作为预设值，因为对于我而言这是我个人需求最多的一条路线。点击起点和终点的下拉选项，可以选择起点和终点，按下键盘上的 Enter 键或者点击导航按钮，都可以触发导航的查找动作。

下面是三次导航查询的例子。

1. 上图是 11 公寓至 H 楼的最短路线。
2. 中图是 M 楼至天雅苑的最短路线。
3. 下图是大学生服务中心至图书馆的最短路线。

点击查询按钮或按下 Enter 键后，在地图显示区域会依据具体的路线，自动画出最短路线并虚线标红。

右边栏也会同步动态显示这次查询的起点，终点和精确的路线距离。



3. 调试分析记录

问题和困难

在软件的开发调试过程中，我遇到最大的问题就是如何让路线能够 UI 显示出来。

为了解决这个问题，我建造了庞大的数据路线点集，将地图上每条道路都标记出来，依次在地图上进行取点，记下关键点。通过对距离的查找，找到对应路线，进而找到对应路线点集。

无向图的点与点间距离，必然需要真实的数据。所以我在高德地图接口，找到了距离的下载入口，将这些路线和距离下载出来，存入了无向图中。

由于对 Qt 的不熟悉，在初期搭建主界面时，也遇到了边栏和地图窗口无法共存，背景颜色无法设定，UI 字体错误等问题，最终通过对 Qt GUI 的深入学习，我找到了两者共存的方法。

运行时间和所需内存

Floyd 的运行时间为 32.2834ms。

Dijkstra 的运行时间为 26.5391ms。

插值查找的运行时间为 2.2547ms。

程序空载运行所需内存为 54576K，运行核心算法需要最大内存为 56124K。

算法和数据结构的改进

1. 散列表

类邻接矩阵可以使用更复杂的逻辑替代，比如使用散列表。这样可以更节约内存，并且提升访问速度。

2. 窗口的缩放功能

在使用此程序时，我发现地图和窗口的大小以及固定，不适合所有用户使用，但任务初期并没有考虑这一点，所以我认为应该加入窗口的缩放功能，将其更完善。

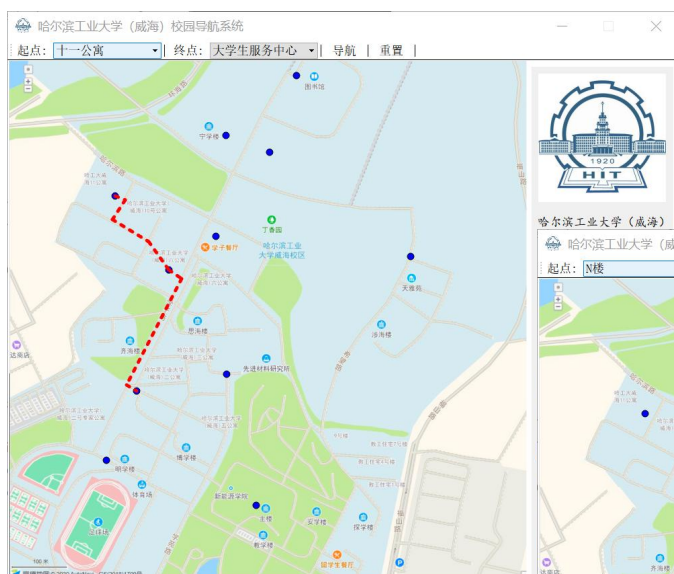
4. 运行结果与分析

1. 短距离路线显示

下图分别为 11 公寓至大学生服务中心的最短路线、N 楼至 8 公寓的最短路线、大学生活动中心至天雅苑的最短路线、学苑餐厅至 H 楼的最短路线。

四者皆为距离较短，路线单一，经过地点为 0 或 1 的数据。

由路线图可见，路线的显示比较精确。

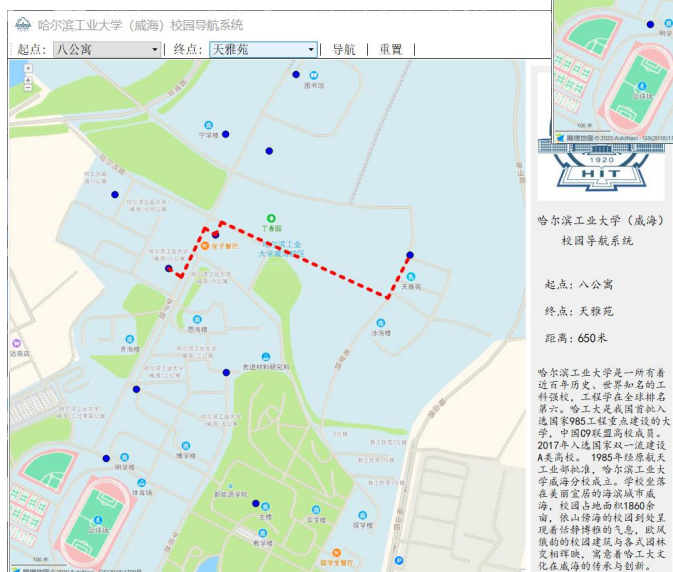


2. 中距离路线显示

上图为学苑餐厅至图书馆的最短路线，中图为 H 楼至天雅苑的最短路线，下图为 8 公寓至天雅苑的最短路线。

三者皆为距离适中，路线多样，经过地点为 1 至 2 个的数据。

此时需要用 Floyd 算法计算出最短路线。

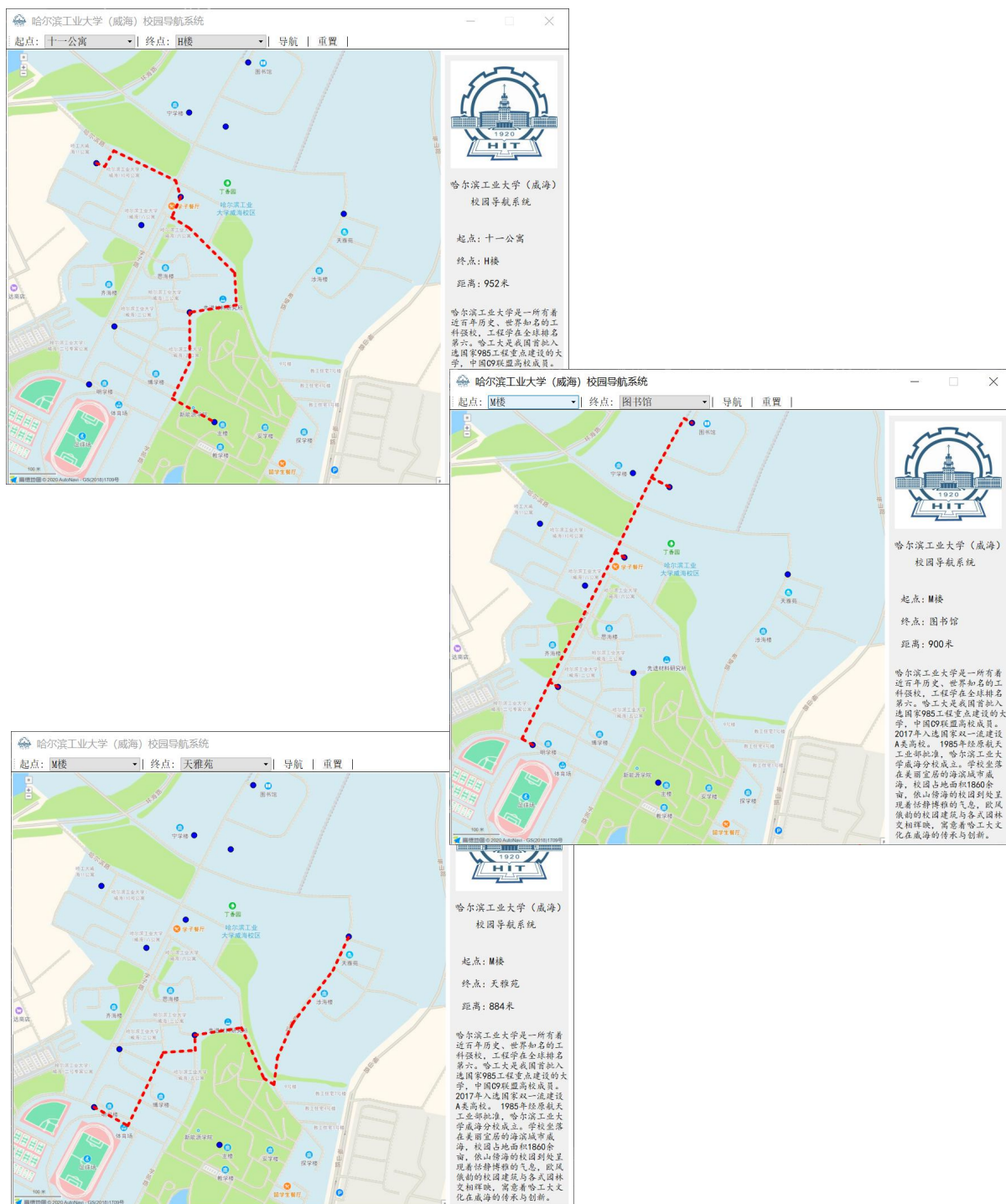


3. 长距离路线显示

上图为 11 公寓至 H 楼的最短路线，中图为 M 楼至图书馆的最短路线，下图为 M 楼至天雅苑的最短路线。

三者皆为距离比较长，路线比较多，经过地点也比较多。

此时需要用 Dijkstra 算法计算出最短路线。



4. 起点终点互换的处理

上图为大学生服务中心至天雅苑的最短路线。

下图为天雅苑至大学生服务中心的最短路线。

互换起点和终点时，显然最短路线和距离应该相同，可以验证内部转换逻辑是正确的。



5. 起点终点相同的处理

下图为当起点和终点选择相同的情况。

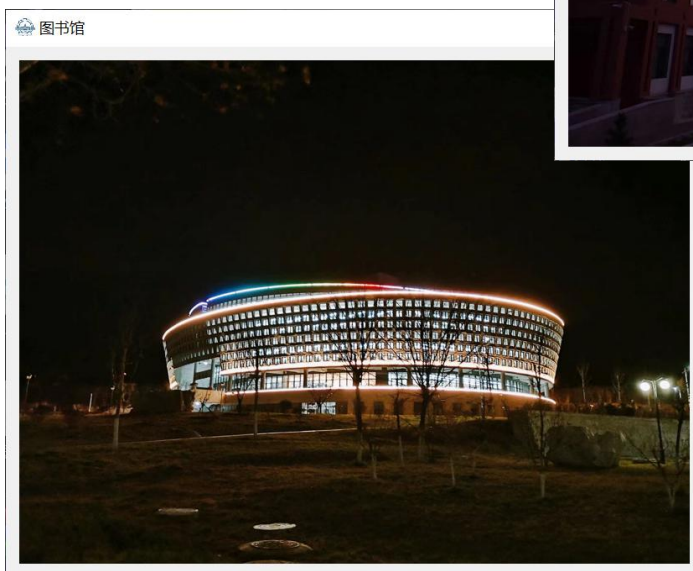
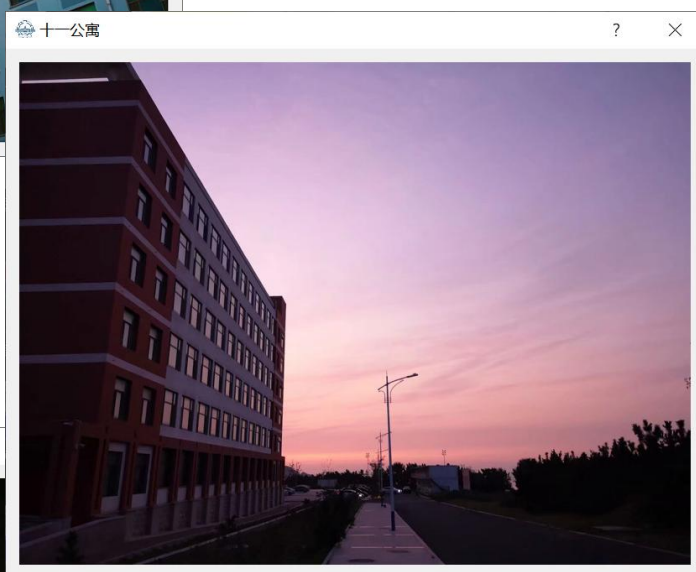
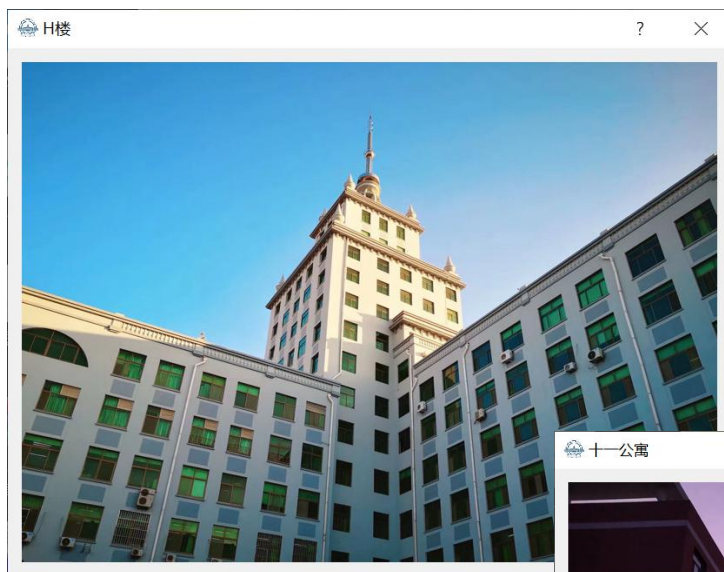
起点和终点选择相同时，因为没有意义，所以不作路线计算。

没有路线画出，并且距离显示为文字“请重新选择”。



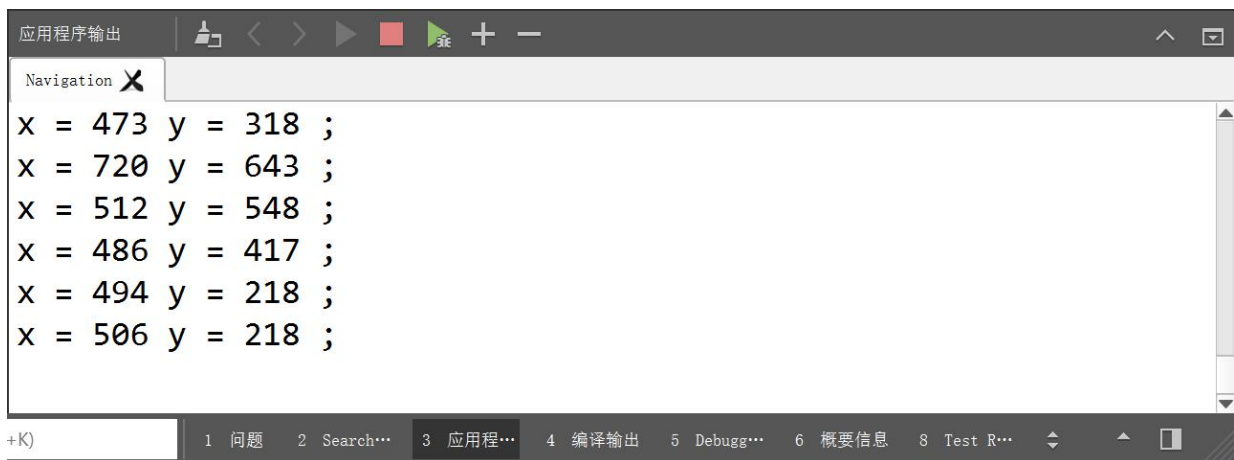
6. 显示地点的图片介绍。

双击图中地点圆点时，会出现一个窗口，展示这个地点的图片介绍，方便客人认识并查找辨认建筑。



7. 控制台显示

1. 点击地图上任意一点时，控制台会输出其坐标。
2. 查找路线时，控制台会输出起点、经过的地点、终点、距离等详细数据。



The screenshot shows the Qt console window with the title '应用程序输出'. The console output displays a series of coordinate pairs (x, y) separated by semicolons, representing points on a map. The points are: (473, 318), (720, 643), (512, 548), (486, 417), (494, 218), and (506, 218). The console window has a toolbar with icons for file operations and a status bar at the bottom showing tabs for '问题', 'Search...', '应用程...', '编译输出', 'Debug...', '概要信息', and 'Test R...'.

```
Navigation X
x = 473 y = 318 ;
x = 720 y = 643 ;
x = 512 y = 548 ;
x = 486 y = 417 ;
x = 494 y = 218 ;
x = 506 y = 218 ;
```



The screenshot shows the Qt console window with the title '应用程序输出'. The console output displays the result of a 'find path' operation, showing a sequence of points and the distance: '4 to 7: 7 to 6 to 8 to 9 to 4 (Dis: 1030)'. Below this, there is a message from QPainter: 'QPainter::begin: Paint device returned engine == 0, type: 1'. The console window has a toolbar and a status bar at the bottom.

```
Navigation X
find path
4 to 7: 7 to 6 to 8 to 9 to 4 (Dis: 1030)
QPainter::begin: Paint device returned engine == 0, type: 1
```



The screenshot shows the Qt console window with the title '应用程序输出'. The console output displays the result of a 'find path' operation, showing a sequence of points and the distance: '0 to 2: 2 to 5 to 1 to 0 (Dis: 812)'. Below this, there is a message from QPainter: 'QPainter::begin: Paint device returned engine == 0, type: 1'. The console window has a toolbar and a status bar at the bottom.

```
Navigation X
find path
0 to 2: 2 to 5 to 1 to 0 (Dis: 812)
QPainter::begin: Paint device returned engine == 0, type: 1
```


5. 教师指导建议及解决记录

5.1 开题检查

老师的指导建议：

1. UI 画图一定要精准可视化，做好位置标注否则计算路径的时候就会影响视觉效果
2. 对每个点加上 一些介绍，如图片，教室的空闲情况介绍等
3. 可以和百度地图等做一些结合

解决措施：

1. 我将每条路线都设定了各自的路线点集，精确到具体的坐标位置，保证 UI 画图的精准。
2. 由于对 Qt 的不熟练和我们学校官网信息的不全面，我没能找到教学楼的概况和介绍，不过我找到了部分我自己拍摄的学校图片，将其设定为双击展示图片。
3. 我利用了高德地图的接口，将高德地图上我们学校的地图打印出，作为地图展示。

效果：

1. UI 画图如上图一样，非常精准，保证了信息完整性。
2. 可以将地点图片进行展示，实现了一部分介绍地点的目的。
3. 地图可以清晰完整的展示出来，但是地图未设置缩放功能。

5.2 中期检查

老师的指导建议：

1. 一定要加快工作进度，保证完整性。
2. 展示出完整的路线 UI。
3. 美化 UI 界面，展示出复杂的功能。

解决措施：

1. 由于我前中期准备不足，中期检查的工作能够展现出的确实比较少，之后我加紧了工作进度。
2. 中期检查时，UI 展示界面很不健全，只有地图可以展示，右边栏和工具栏仍然没有编写好，后期加快了工作进度。
3. 中期检查时 UI 界面很少，所以老师也希望我的 UI 能更加美观。后期我确定了我设计 UI 的设计基调，保持简洁易上手。

效果：

1. 终检时完成了程序的所有功能。
2. 可以完整的演示地图上所有地点间的最短路线查询和展现。
3. 最终程序的 UI 正如我想象中的一样，保持外观的整洁，以白色和浅灰为主基调，搭配校徽和黑色字体，查找功能明显易用。

5.3 软件验收

老师的指导建议：

1. 我的程序相对中期检查来讲，工作量大了很多，程序完善了很多，功能基本完全实现，并且保持美观。
2. 对于地点的介绍，可以多添加一些内容，例如教学楼的教室情况，座位数量，建造时间等等。
3. 如若可能，将地点介绍与教务系统联网对接，直接查询，这样效果会更好。

6. 总结

这次的软件设计与开发实践课程，经过了纠结万分的选题，困难重重的设计和编程，最终功夫不负有心人，我完成了一个自己还算满意的工艺品——哈尔滨工业大学（威海）校园导航系统。

这次的课程设计，我收获了很多。在开题时，我曾经很迷茫，不知道如何将数据结构和算法应用到实际。课程刚发布时，我不太理解这门课程的意义，以为只是再一次用一些技术，完成一个程序。但是开题时我发现我理解错了，这门课是专于运用数据结构和算法，帮助我们彻底理解的。之后在构思校园导航系统时，我想了很多，顺序表、散列表、有向图、无向图、邻接表，我逐渐意识到数据结构的复杂和用途。我认为数据结构和算法是不可分割的，数据结构为算法提供高效的基础，算法使数据结构发挥真正的用途。

我个人认为，这次软件设计与开发实践，我实现了很多功能，将路线 UI 做的很精准，我是比较满意的，但我做的并不是完美的。第一，路线 UI 的调试和添加，是使用坐标点在这个地图上找出的，若加入地图缩放功能，这种方式可能会有一定局限性。第二，路线点集的存储使用的是邻接矩阵，但我后来经过思考，我认为若使用散列表，可能会更完美，因为其并不是满的邻接矩阵，使用散列表或哈希表会节省很大一部分空间，让我的算法得到很大程度的优化。

经过这次开发实践，我充分的意识到，其实我的能力是有限的。我可能在某些方便比较优秀，但我还有很多需要学习，提升自己的地方。我会在以后的学习道路上，不断用实践提升自我。