组号: 8


上海大学计算机工程与科学学院

# 实 验 报 告

## （数据结构 2）

学　　期：春

组　　长：邱姜铭

学　　号：22122861

指导教师：朱频频

成绩评定：（教师填写）

二〇二三年四月六日

| 小组信息 | | | | |
|---|---|---|---|---|
| 登记序号 | 姓名 | 学号 | 分工 | 签名 |
| 1 | 邱姜铭 | 22122861 | 组长 | |
| 2 | 向铮皓 | 22122867 | ppt 和报告 | |
| 3 | 申帅男 | 22122072 | ppt 和报告 | |
| 4 | 唐天澄 | 19122405 | 界面 | |
| 5 | 张欣悦 | 22122796 | 界面 | |
| 6 | 王萌 | 22121876 | 算法 | |
| 7 | 徐王嘉 | 22122748 | 算法 | |
| 8 | 曹悦昕 | 22121027 | 测试 | |

| 实验概述 | |
|---|---|
| 实验零 | （熟悉上机环境、进度安排、评分制度；确定小组成员） |
| 实验一 | 家谱管理系统 |
| 实验二 | 乡村公路建设与乡村医院规划 |

# 实验二

## 一、实验题目

乡村公路建设与乡村医院规划

## 二、实验内容

### 1、任务目标

你们小组作为上海大学派出的大学生团队，帮助西北某乡进行乡村公路和乡村医院建设规划的工作。乡村公路线长面广，分散在各个地区的各个角落，另外由于地形复杂，并不是所有村之间都可以直接建设公路。为了节省资金，需要用最经济的方案完成乡村公路建设和乡村医院的规划。

### 2、任务描述

任务有以下要求：

1）要求给出用最少资金就能使所有村都能通路的方案；

2）假设所有能建的公路都已经建成。需要确定乡村医院造在哪个村，可以使所有村到该医院的总路程最短；

3）为了也能帮助其他乡完成类似工作，需要设计一个完整的程序，有交互界面，有文件导入等功能，详见下述功能列表；

### 3、功能列表

1）导入指定的文件，显示相应的图，显示方式自定；

2）完成任务 1，给出最少资金方案，展示方式自定；

3）完成任务 2，给出医院设置点和对应的总路程，展示方式自定；

## 三、解决方案

### 1、算法设计分析

在本实验中使用 C++语言进行编程。根据实验要求，我们小组自行设计了并查集类，无向图类等，并且基于 Qt 框架的图形界面应用程序完成了交互的功能，下面将详细阐述各部分完成的功能。

(1) 并查集 UnionFind

```
class UnionFind {
private:
    int *parent;
    int *rank;
public:
    explicit UnionFind(int n) {
        parent = new int[n];
        rank = new int[n];
        for (int i = 0; i < n; i++) {
            parent[i] = i;
            rank[i] = 0;
        }
    }
}
```

在上面这段代码中，UnionFind 类有两个私有成员变量：parent 和 rank。parent 数组用于存储每个元素的父节点，rank 数组用于记录每个根节点的秩。

构造函数 UnionFind(int n)用于初始化并查集，参数 n 表示并查集的大小。在构造函数中，通过循环遍历初始化 parent 和 rank 数组，将每个元素的父节点设置为自己，秩初始化为 0。

```
    int find(int x) {
        if (x != parent[x]) {
            parent[x] = find(parent[x]);
        }
        return parent[x];
    }
```

find(int x)函数用于查找元素 x 所在的集合的根节点。如果 x 不是根节点，则通过递归调用 find 函数来路径压缩，将 x 的父节点更新为根节点，并返回根节点。

```
    void merge(int x, int y) {
        int rootX = find(x);
        int rootY = find(y);
        if (rootX == rootY) {
            return;
        }
        if (rank[rootX] < rank[rootY]) {
            parent[rootX] = rootY;
        } else if (rank[rootX] > rank[rootY]) {
            parent[rootY] = rootX;
        } else {
            parent[rootX] = rootY;
            rank[rootY]++;
        }
    }
```

merge(int x, int y)函数用于合并元素 x 和 y 所在的两个集合。首先找到 x 和 y 的根节点 rootX 和 rootY，如果 rootX 和 rootY 相等，则表示 x 和 y 已经在同一个集合中，直接返回。如果 rootX 的秩小于 rootY 的秩，则将 rootX 的父节点设置为 rootY；如果 rootX 的秩大于 rootY 的秩，则将 rootY 的父节点设置为 rootX；如果 rootX 的秩等于 rootY 的秩，则将 rootX 的父节点设置为 rootY，并将 rootY 的秩加 1。

```
    bool connected(int x, int y) {
        return find(x) == find(y);
    }
```

connected(int x, int y)函数用于判断元素 x 和 y 是否在同一个集合中。通过调用 find 函数找到 x 和 y 的根节点，如果根节点相同，则表示 x 和 y 在同一个集合中，返回 true，否则返回 false。

整个代码实现了并查集的基本操作，包括初始化、查找、合并和判断连通性。它可以用于解决一些与集合合并和连通性相关的问题，例如判断网络中的节点是否连通、求解最小生成树等。本实验中该类用于辅助实现克鲁斯克尔算法。

(2) 无向图 Graph

```
template<class W, class T>
class Graph {
public:
    // 权重类型
    typedef W weight_type;
    // 数据类型
    typedef T data_type;
    // 顶点类型
    typedef struct Point {
        data_type name;
        int x;
        int y;
    } point_type;
    // 边类型
    typedef pair<weight_type, int> edge_type;
```

首先，代码定义了三个类型别名：weight_type，data_type 和 point_type。weight_type 用于表示边的权重类型，data_type 用于表示顶点的数据类型，point_type 用于表示顶点的结构类型。point_type 结构包含了顶点的名称（name）以及其在二维坐标系中的坐标（x 和 y）。

```
private:
    // 顶点数
    int n;
    // 边数
    int m;
    // 邻接表 adj[u] = {v1, v2, ...}
    vector<vector<edge_type>> adj;
    // 顶点名
    vector<point_type> points;
```

接下来，代码声明了私有成员变量：n、m、adj 和 points。n 表示图中顶点的数量，m 表示图中边的数量。adj 是一个二维向量，用于存储邻接表。adj[u] 表示顶点 u 的邻接顶点列表，每个邻接顶点由一个 pair 组成，包含了边的权重

和邻接顶点的编号。points 是一个向量，用于存储顶点的信息。每个顶点信息由 point_type 结构表示，包含了顶点的名称和坐标信息。

```
    void addEdge(int u, int v, weight_type w);

    vector<point_type> &getPoints();

    vector<pair<int, int>> getEdges();

    bool hasEdge(int u, int v);

    weight_type getWeight(int u, int v);

    [[nodiscard]] int V() const;

    [[nodiscard]] int E() const;
```

这个 Graph 模板类可以用于表示不同类型的图，其中顶点的数据类型和权重的类型可以根据实际需要进行指定。它提供了邻接表表示法来存储图的结构，并提供了一些基本的操作和访问方法，例如获取顶点数量、边数量、顶点的邻接顶点等。

(3) 克鲁斯克尔算法与弗洛伊德算法

```
    //克鲁斯克尔算法
    Graph<W, T> *kruskal() {
        // 生成最小生成树
        auto tree = new Graph<W, T>(points);
        // 边集
        vector<pair<W, pair<int, int>>> edges;
        for (int u = 0; u < n; u++) {
            for (auto &edge: adj[u]) {
                int v = edge.second;
                if (u < v) {
                    edges.emplace_back(edge.first,
std::make_pair(u, v));
                }
            }
        }
        // 按权重排序
        std::sort(edges.begin(), edges.end());
        // 并查集
        UnionFind uf(n);
```

```
        // 遍历边集
        for (auto &edge: edges) {
            int u = edge.second.first;
            int v = edge.second.second;
            // 如果 u 和 v 不连通
            if (!uf.connected(u, v)) {
                // 添加边
                tree->addEdge(u, v, edge.first);
                // 合并 u 和 v
                uf.merge(u, v);
            }
        }
        return tree;
    }
```

克鲁斯克尔算法用于生成最小生成树。首先，创建一个新的图对象 tree，用于存储最小生成树。然后，将图中的边按照权重从小到大进行排序。接下来，使用并查集数据结构来判断边的两个顶点是否连通。遍历边集，如果两个顶点不连通，则将边添加到最小生成树中，并将这两个顶点合并到同一个连通分量中。最后，返回最小生成树。

```
    // 弗洛伊德算法
    vector<vector<weight_type>> floyd() {
        // 无穷大
        const auto weight_inf =
std::numeric_limits<weight_type>::max();
        // 距离矩阵
        vector<vector<weight_type>> dist(n,
vector<weight_type>(n, weight_inf));
        // 初始化
        for (int i = 0; i < n; i++) {
            dist[i][i] = 0;
        }
        for (int u = 0; u < n; u++) {
            for (auto &edge: adj[u]) {
                int v = edge.second;
                dist[u][v] = edge.first;
            }
```

```
        }

        // 弗洛伊德算法
        for (int k = 0; k < n; k++) {
            for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++) {
                    if (dist[i][k] != weight_inf &&
                        dist[k][j] != weight_inf) {
                        dist[i][j] = std::min(dist[i][j],
dist[i][k] + dist[k][j]);
                    }
                }
            }
        }
        return dist;
    }
```

弗洛伊德算法用于计算所有顶点对之间的最短路径。首先，创建一个距离矩阵 dist，用于存储顶点之间的最短路径距离。将所有距离初始化为无穷大，然后将已知的边的权重添加到距离矩阵中。接下来，使用三重循环来遍历所有顶点对。如果经过顶点 k 的路径比直接连接更短，则更新最短路径距离。最后，返回距离矩阵。

(4) 图的展示 GraphView

```
class GraphView : public QWidget {
Q_OBJECT

private:
    typedef Graph<int, std::wstring> graph_type;

public:
    explicit GraphView(QWidget *parent = nullptr);

    ~GraphView() override;

    // 从文件读取
    void readFromFile(const std::string &filename);

    // 最小生成树
    void minimumSpanningTree();
```

```
    // 最短路径
    void shortestPath();

    // 重置最小生成树
    void resetMinimumSpanningTree();

    // 重置最短路径
    void resetShortestPath();

protected:
    void paintEvent(QPaintEvent *event) override;

    void mousePressEvent(QMouseEvent *event) override;

    void mouseReleaseEvent(QMouseEvent *event) override;

    void mouseMoveEvent(QMouseEvent *event) override;

};
```

以上代码中，在构造函数中，首先调用 ui->setupUi(this)初始化 UI 界面，然后设置背景颜色为白色。

paintEvent(QPaintEvent *event)函数用来重写了 QWidget 的绘图事件处理函数，用于绘制图形数据结构中的点和边。在函数中，首先获取绘图对象 QPainter，设置抗锯齿和画笔颜色，然后绘制点和边，根据点的状态（选中、最短路径点等）设置不同的颜色和样式。

mousePressEvent(QMouseEvent *event)函数重写了鼠标按下事件处理函数，用于处理鼠标点击事件。在函数中，根据鼠标点击位置判断是否选中了某个点，并记录选中点的索引。

mouseReleaseEvent 函数重写了鼠标释放事件处理函数，用于处理鼠标释放事件。在函数中，重置选中点的索引，并更新界面。

mouseMoveEvent(QMouseEvent *event)函数重写了鼠标移动事件处理函数，用于处理鼠标移动事件。在函数中，判断选中点是否在边界内，更新选中点的位置，并更新界面。

readFromFile 函数从文件中读取数据，包括国家名和边的信息，构建图形数据结构，并设置点的位置

minimumSpanningTree 函数计算最小生成树，并更新显示。

shortestPath 函数计算所有点的最短路径，找到最短路径的起点和总权重，并更新显示。

resetMinimumSpanningTree 和 resetShortestPath 函数用于重置最小生成树和最短路径的状态。

这些函数实现了在图形界面中显示图形数据结构的功能，并提供了交互操作，如选择点、移动点等，通过 Qt 的信号和槽机制与界面交互，实现了图形数据结构的可视化和操作功能。

(5) 主界面类 MainView

```
class MainView : public QWidget {
Q_OBJECT

public:
    explicit MainView(QWidget *parent = nullptr);

    ~MainView() override;

private:

    void chooseFile();

    void readFromFile();

    void minTree();

    void shortestPath();

};
```

在构造函数中，使用 ui->setupUi(this)来设置界面的布局和组件。然后通过 this->setStyleSheet("background-color: white;")来设置窗口的背景颜色为白色。接着，通过 QObject::connect 函数将按钮的点击事件与对应的槽函数进行绑定，readFromFile 槽函数用于读取文件，minTree 槽函数用于生成最小生成树，shortestPath 槽函数用于计算最短路径，chooseFile 槽函数用于选择文件。

chooseFile 函数用于选择文件，使用 QFileDialog::getOpenFileName 函数弹出文件选择对话框，选择文件后将文件名保存在_filename 变量中，并将文件名显示在选择文件按钮上。

readFromFile 函数用于从文件中读取图的数据。如果_filename 为空，则弹出错误对话框提示用户先选择文件。否则，调用 ui->graphView->readFromFile(_filename)将文件的数据传递给图形视图对象 graphView 进行处理。

minTree 函数用于生成最小生成树。使用一个静态变量 flag 来记录当前是否已经生成了最小生成树。如果 flag 为 false，则调用 ui->graphView->minimumSpanningTree()生成最小生成树，并将按钮的文本设置为"重置"，flag 设为 true；如果 flag 为 true，则调用 ui->graphView->resetMinimumSpanningTree()重置最小生成树，并将按钮的文本设置为"最小生成树"，flag 设为 false。

shortestPath 函数用于计算最短路径。同样使用一个静态变量 flag 来记录当前是否已经计算了最短路径。如果 flag 为 false，则调用 ui->graphView->shortestPath()计算最短路径，并将按钮的文本设置为"重置"，flag 设为 true；如果 flag 为 true，则调用 ui->graphView->resetShortestPath()重置最短路径，并将按钮的文本设置为"最短总路径"，flag 设为 false。

这段代码实现了一个具有文件选择、最小生成树和最短路径计算功能的图形界面应用程序的主界面类。用户可以通过选择文件来读取图的数据，然后可以生成最小生成树和计算最短路径，并且可以在生成和重置之间切换。

(6) 处理字符串

```
#include <string>
#include <codecvt>

const std::locale utf8_locale = std::locale(std::locale(), new
std::codecvt_utf8<wchar_t>);
const std::wstring DELIMITER = L"==========我是分割线
==============";
```

首先，代码定义了一个常量 std::locale utf8_locale，它是一个用于处理 UTF-8 编码的本地化对象。该对象使用了 std::codecvt_utf8<wchar_t>来实现 UTF-8 和宽字符之间的转换。

接下来，代码定义了一个常量 std::wstring DELIMITER，它是一个宽字符串常量，用于表示一个分割线。这个分割线可以在字符串处理中作为一个标记，用于分隔不同的内容。

这段代码主要用于辅助处理中文字符编码和读取文件。

(7) ui_mainview 与 ui_graphview

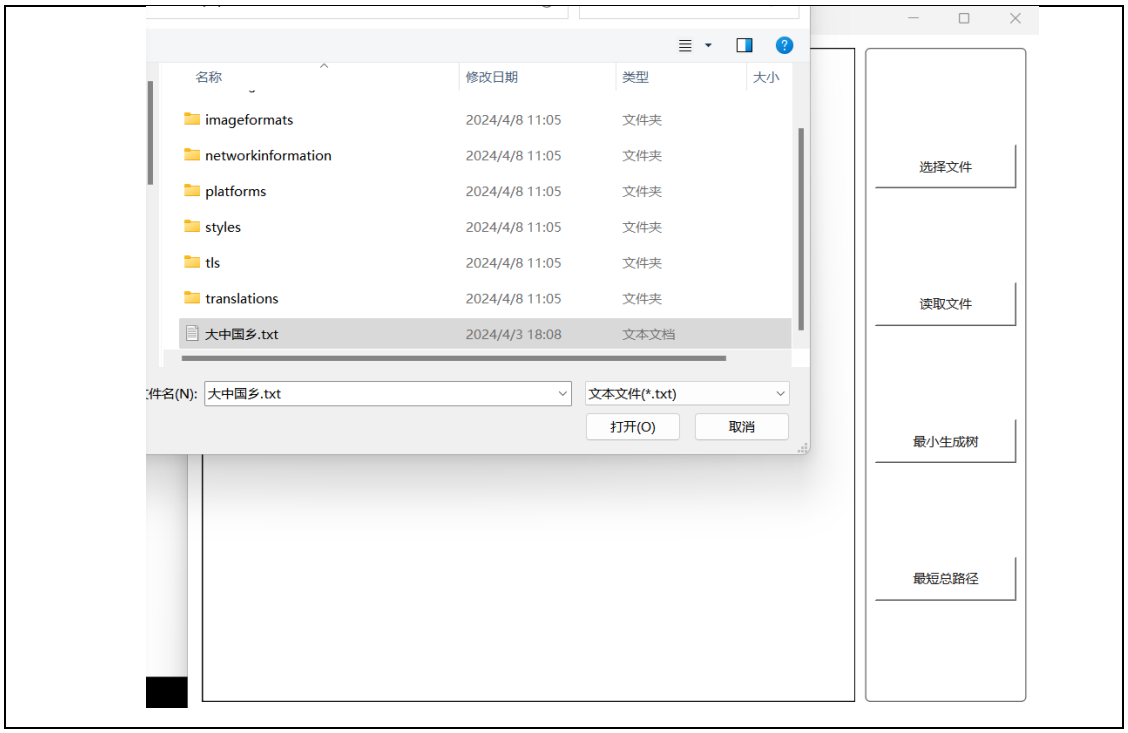
这两个文件由.ui 文件自动生成，这里不做代码展示，主要是定义了一个名为 MainView 的 QWidget 界面，包含了一个 GraphView 对象和一些按钮，用于展示图形数据结构的相关操作。该界面的布局由水平布局和网格布局组成，其中水平布局包含了 GraphView 对象和一个 QGroupBox，QGroupBox 中包含了四个 QPushButton 按钮，分别用于选择文件、读取文件、计算最小生成树和计算最短路径。整个界面的布局和按钮事件的绑定等操作都在该 UI 设计文件中完成。

## 2、结果展示

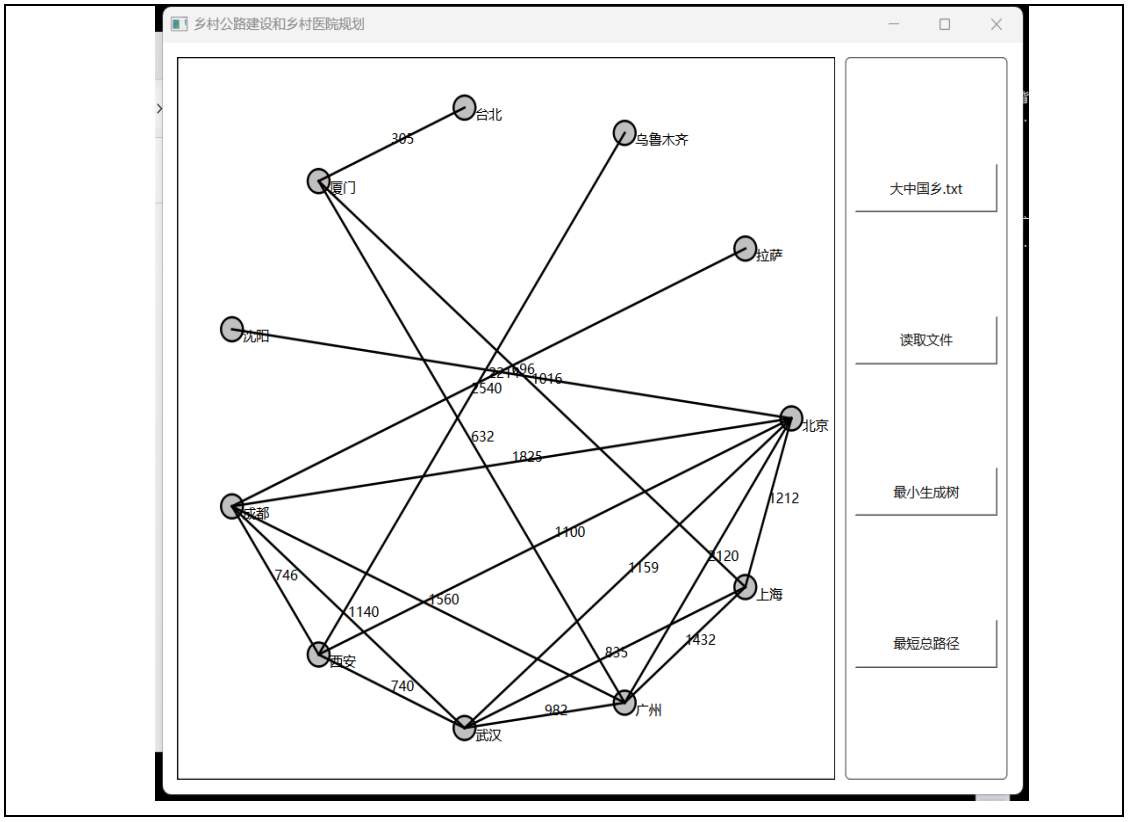
本实验的各个功能的展示效果如下：

有四个功能选项：分别为选择文件、读取文件、最小生成树、最短总路径

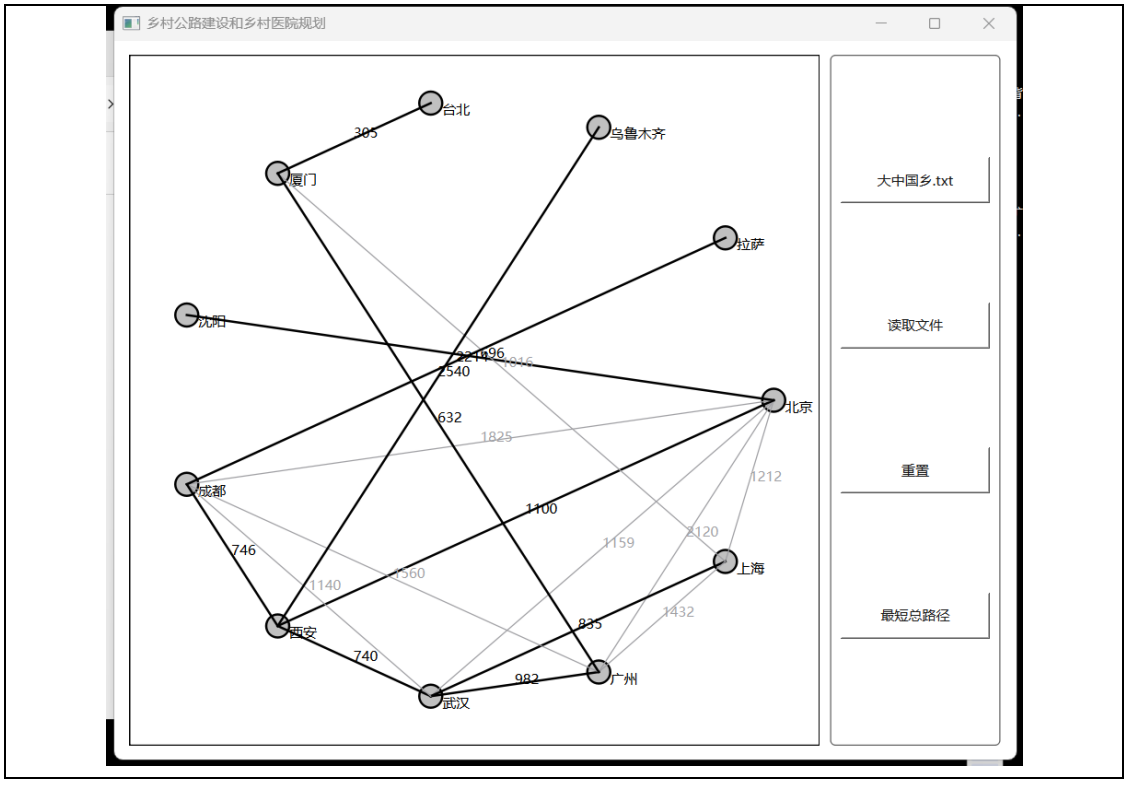

我们选取“大中国乡.txt”文件



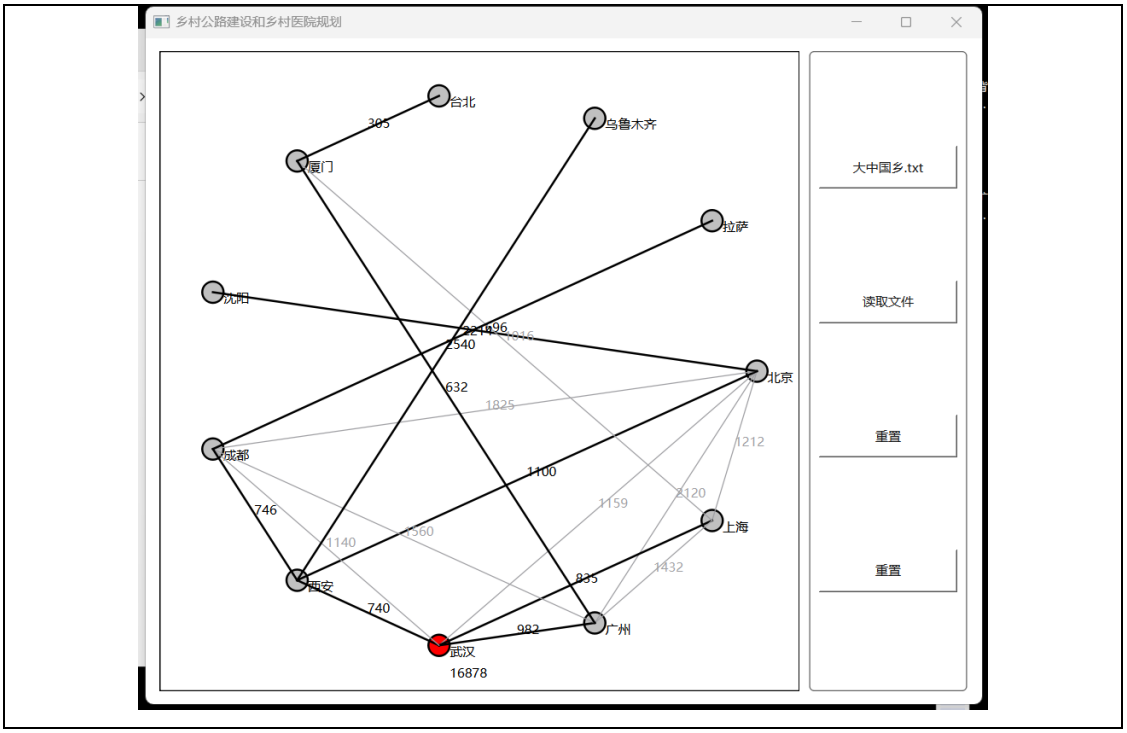
读取该文件，如图所示：



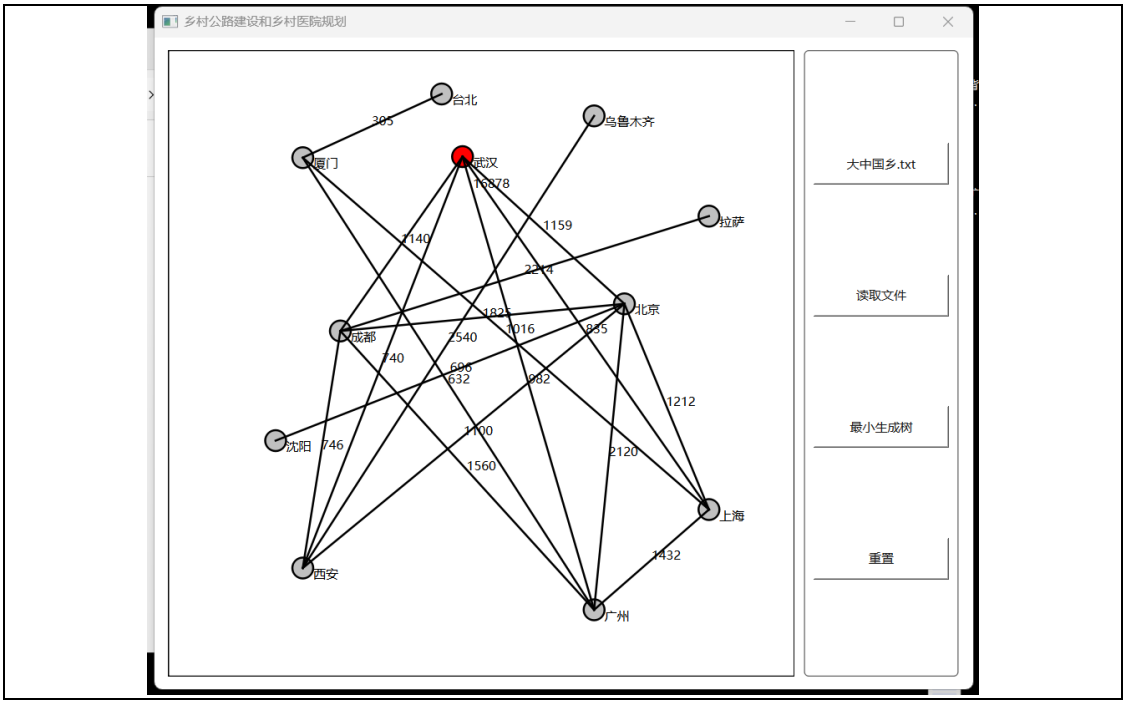

最小生成树：



最短总路径：




我们还设置了可以移动图形中各点的位置，可以在新的图形上重复以上操作：



## 3、总结与心得

在本次作业中，我们采用了 Kruskal 算法来找到连接所有村庄的最小成本公路网络，而使用了 Floyd 算法来计算最短总路径。关于表示图结构的数据结构，我们使用了邻接表，以便在 C++中有效地表示和操作图。算法的设计考虑了以下几个方面：

我们深入研究了 Prim 算法和 Kruskal 算法，并根据问题的特点选择了适合的算法。在我们的情况下，由于需要找到一个连接所有村庄的最小成本网络，我们选择了 Kruskal 算法，因为它更适合在边稀疏的图中找到最小生成树。

而关于最短总路径，我们采用 Floyd 算法是因为它能够同时找出所有节点之间的最短路径，而不仅仅是单个节点到其他节点的最短路径，同时实现简单，使得我们最后采用了该算法。

在展示图形界面方面，我们使用了 Qt 框架，设计了直观友好的用户界面，包括以下几点：

交互设计：我们设计了简洁清晰的交互界面，使用户可以轻松导入数据、执行算法、查看结果。结果可视化：通过图形化展示，我们将最小生成树和最短路径等结果直观地呈现给用户，提高了用户对结果的理解和认识。用户反馈：我们考虑了用户体验，为程序添加了一些友好的提示和错误处理，使用户在使用过程中更加流畅和愉快。

在 C++中处理中文字符可能会遇到编码、输入输出等问题，我们的处理方式如下：

编码选择:我们统一使用了 UTF-8 来存储代码文件和数据文件。字符串存储：我们统一使用了 std::wstring 来存储，并在不同地方使用了不同的转换方式来对于不同情况进行适配。如读取文件时使用了 std::codecvt_utf8，在图像化显示时使用 QString::fromStdWString 等。

通过本次作业，我们小组不仅学习了图算法在实际问题中的应用，还掌握了使用 Qt 框架设计和实现图形界面的方法。同时，在 C++中处理中文字符的经验也为我们以后的开发工作提供了宝贵的参考。这次作业不仅是对课程知识的巩固，也提升了我们团队合作和问题解决能力，是一次非常有意义的实践活动。