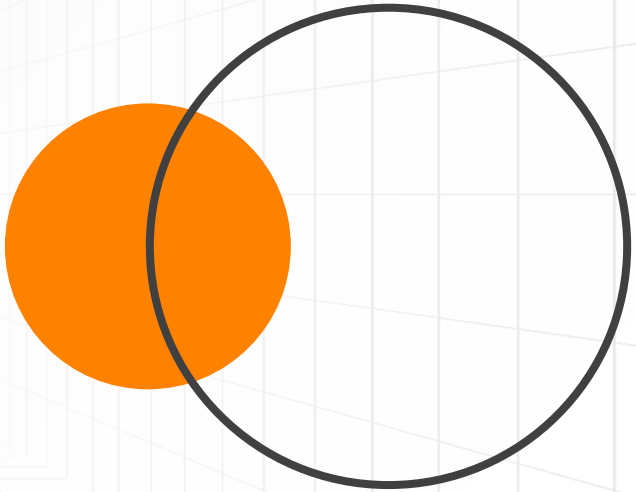
主题乡村公路建设与乡村医院规划

汇报人：第八组

Catalogue 目录

01

章节1 实验概述



子章节1.1 实验信息

组号 8

成员列表及分工：

邱姜铭 (组长)
向铮皓 (ppt 和报告)
申帅男 (ppt 和报告)
唐天澄 (界面)
张欣悦 (界面)
王萌 (算法)
徐王嘉 (算法)
曹悦昕 (测试)

子章节1.2 实验内容

01

任务目标

1、帮助西北某乡进行乡村公路和乡村医院建设规划
2、用最经济的方案完成乡村公路建设和乡村医院的规划

02

任务描述

1、要求给出用最少资金就能使所有村都能通路的方案
2、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、设计一个完整的程序，有交互界面，有文件导入等功能

03

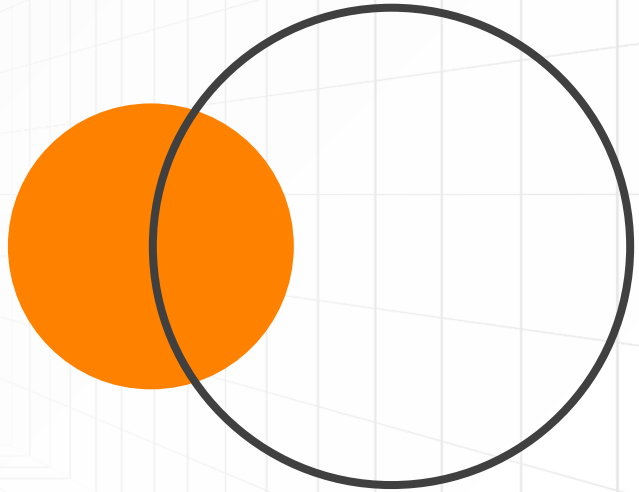
功能列表

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

YOUR LOGO

02

章节2 解决方案



子章节2.1 算法设计分析

01

并查集 UnionFind

UnionFind 类的构造函数用于初始化并查集
find 函数用于查找元素所在的集合的根节点
merge 函数用于合并两个集合
connected 函数用于判断两个元素是否在同一个集合中

02

无向图 Graph

克鲁斯克尔算法用于生成最小生成树
弗洛伊德算法用于计算所有顶点对之间的最短路径

03

克鲁斯克尔算法与弗洛伊德算法

Graph 模板类用于表示不同类型的图
定义了顶点和边的结构类型
提供了获取顶点和边的数量的函数
提供了添加边和获取边权重的函数

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;
        }
    }
```

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。

并查集 UnionFind——find函数

```
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。

并查集 UnionFind——merge函数

```
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]++;
    }
}
```

并查集 UnionFind——connected函数

```
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。

2、无向图 Graph

```
// 无向图
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;
```

2、无向图 Graph

```
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;
public:
    Graph() : n(0), m(0) {}

    explicit Graph(vector<data_type> data) : n(data.size()), m(0), adj(n), points(n) {
        for (int i = 0; i < n; i++) {
            points[i].name = data[i];
        }
    }
```

2、无向图 Graph

```
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) ;

// 返回顶点数
[[nodiscard]] int V() const ;

// 返回边数
[[nodiscard]] int E() const ;
```

3、克鲁斯克尔算法

```
//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));
            }
        }
    }
```

3、克鲁斯克尔算法

```
// 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;
```

3、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;
        }
    }
```

3、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;
```

子章节2.1 算法设计分析

04

图的展示 GraphView

利用一系列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。

05

主界面类 MainView

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。通过选择文件来读取图的数据，生成最小生成树和计算最短路径，并且可在生成和重置之间切换。

06

处理字符串

用于辅助处理中文字符编码和读取文件。

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();
```

4、图的展示 GraphView

```
// 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;
```

4、图的展示 GraphView

```
// 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;
```

4、图的展示 GraphView

```
void GraphView::paintEvent(QPaintEvent *event) {
    QPainter painter(this);
    painter.setRenderHint(QPainter::Antialiasing, true);
    painter.setPen(QPen(Qt::black, 2));
    // 设置边框
        painter.drawRect(0, 0, width(), height());
    if (_graph == nullptr) {
        return;
    }
    auto points = _graph->getPoints(); // 绘制点
```

4、图的展示 GraphView

```
for (auto &point: points) {
    // 判断当前点是否选中
        const bool selected = _selected != -1 && point.name == points[_selected].name;
    // 判断当前点是否为最短路径点
        const bool shortestPath = _shortestPathPoint && point.name == _shortestPathPoint->name;
    if (shortestPath) {
                painter.setBrush(Qt::red); // 如果是最短路径点，设置为红色
    } else if (selected) {
               painter.setBrush(Qt::blue); // 如果是选中点，设置为蓝色
    } else {
                painter.setBrush(Qt::lightGray); // 否则设置为灰色
    }
    // 绘制点
     painter.drawEllipse(point.x - POINT_RADIUS, point.y - POINT_RADIUS, 2 * POINT_RADIUS, 2 * POINT_RADIUS);
    painter.setBrush(Qt::NoBrush);
    // 绘制点的名字
     painter.drawText(point.x + 10, point.y + 10, QString::fromStdWString(point.name));
    if (shortestPath) {
        // 如果是最短路径点，绘制最短路径的总权重
                painter.drawText(point.x + 10, point.y + 30, QString::number(_shortestPathWeight));
    }
}
```

4、图的展示 GraphView

```
// 绘制边
auto edges = _graph->getEdges();
for (auto &edge: edges) {
    // 判断当前边是否为最小生成树的边
        if (_minSpanningTree && !_minSpanningTree->hasEdge(edge.first, edge.second)) {
        // 如果不是最小生成树的边，设置为灰色
                painter.setPen(QPen(Qt::gray, 1));
    } else {
        // 否则设置为黑色
                painter.setPen(QPen(Qt::black, 2));
    }
    painter.drawLine(points[edge.first].x, points[edge.first].y, points[edge.second].x,
points[edge.second].y);
    painter.drawText((points[edge.first].x + points[edge.second].x) / 2,
                (points[edge.first].y + points[edge.second].y) / 2,
                QString::number(_graph->getWeight(edge.first, edge.second)));
}
```

5、主界面类 MainView

```
class MainView : public QWidget {
Q_OBJECT

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

    ~MainView() override;

private slots:

    void chooseFile();

    void minTree();

    void shortestPath();

    void readFromFile();
```

5、主界面类 MainView

```
void GraphView::readFromFile(const std::string &filename) {
    using namespace std;
    wfstream file;
    // 设置文件编码
    file.imbue(utf8_locale);
    file.open(filename, ios::in);
    wstring line;
    vector<wstring> country_names;

    // 读取国家名
        while (file >> line) {
        if (line == DELIMITER) {
            break;
        }
        country_names.push_back(line);
    }
```

5、主界面类 MainView

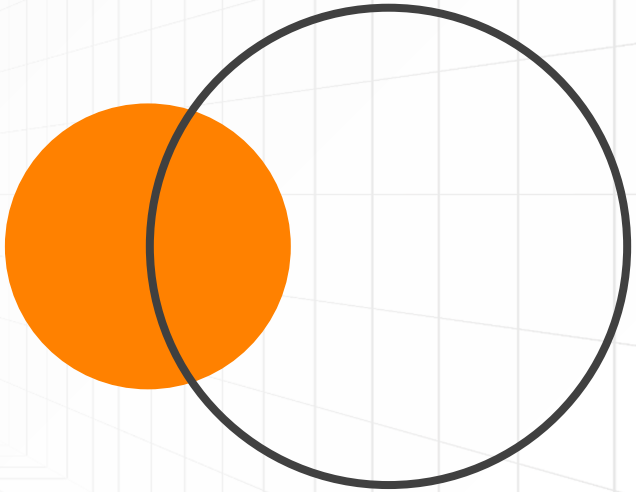
```
auto graph = new Graph<int, wstring>(country_names);

// 读取边
wstring u, v;
int w;

while (file >> u >> v >> w) {
    const auto &_begin = country_names.begin();
    const auto &_end = country_names.end();
    graph->addEdge(distance(_begin, std::find(_begin, _end, u)),
                   distance(_begin, std::find(_begin, _end, v)),
                   w);
}
```

5、主界面类 MainView

```
// 设置点的位置
// 以圆心为中心，均匀分布
const double delta = 2 * M_PI / graph->V();
QPoint center(width() / 2, height() / 2);
for (int i = 0; i < graph->V(); i++) {
    graph->getPoints()[i].x = center.x() + cos(i * delta) * (center.x() - MARGIN);
    graph->getPoints()[i].y = center.y() + sin(i * delta) * (center.y() - MARGIN);
}
_graph = graph;
update();
```

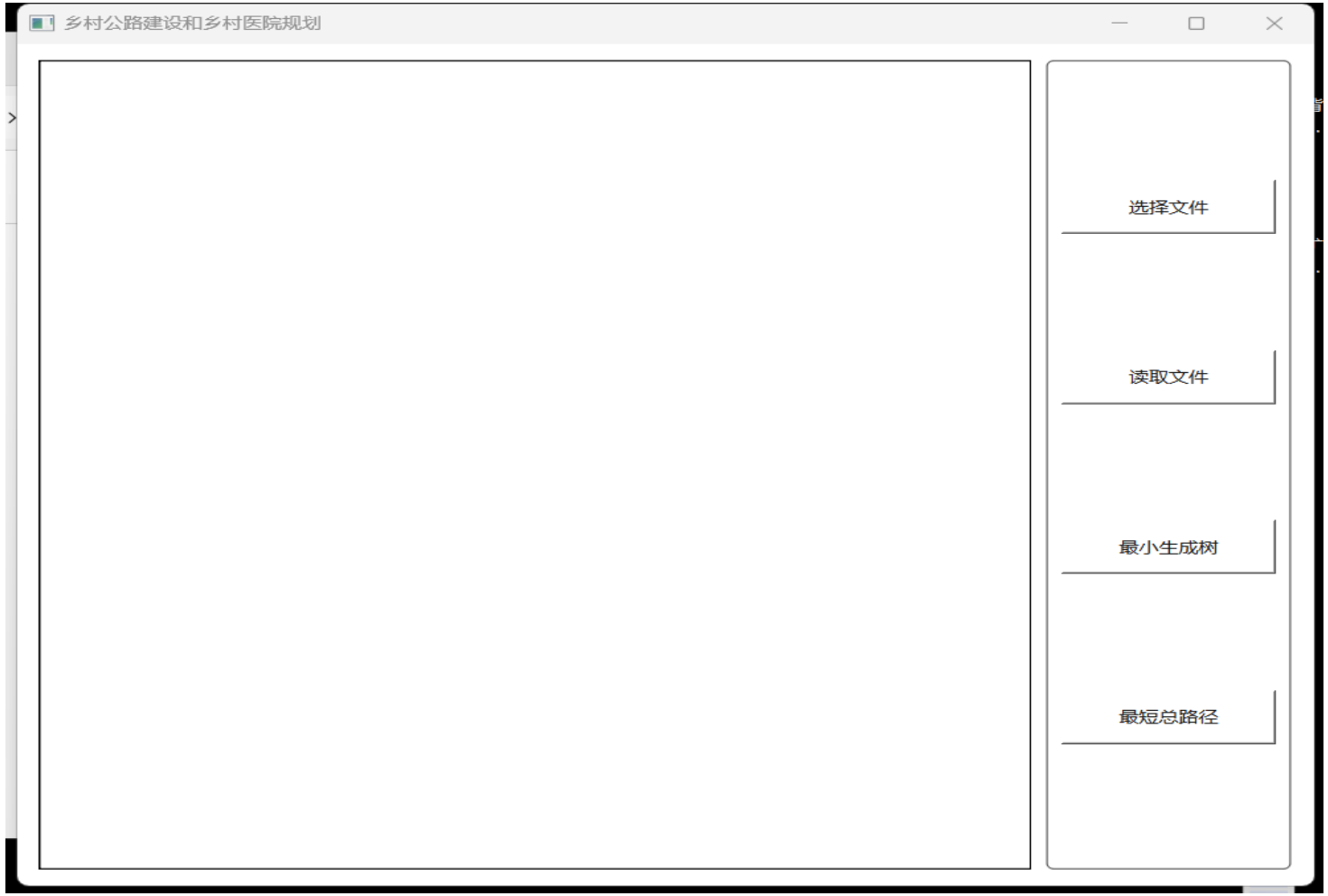
03

章节3 结果展示



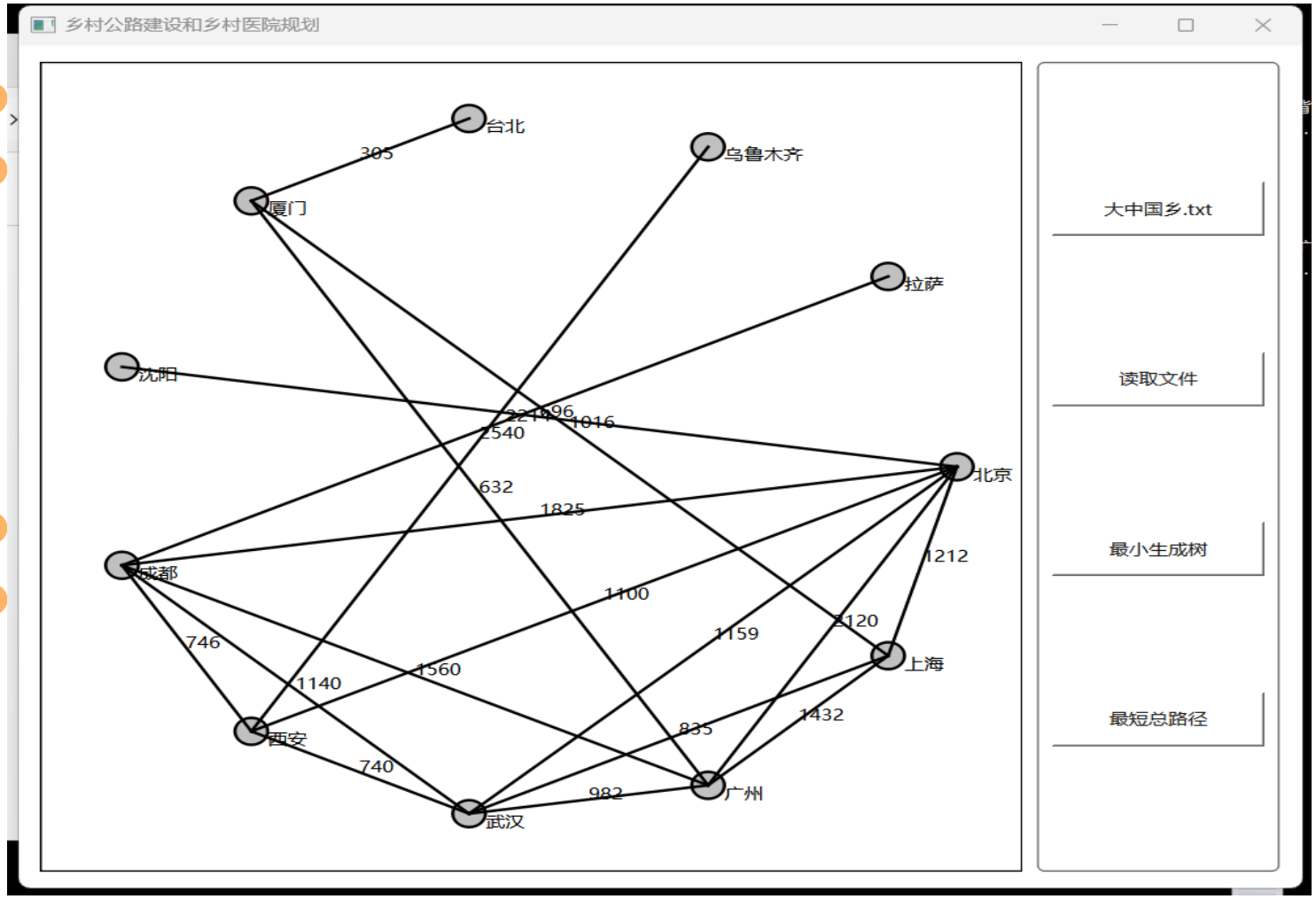
四个功能展示

01



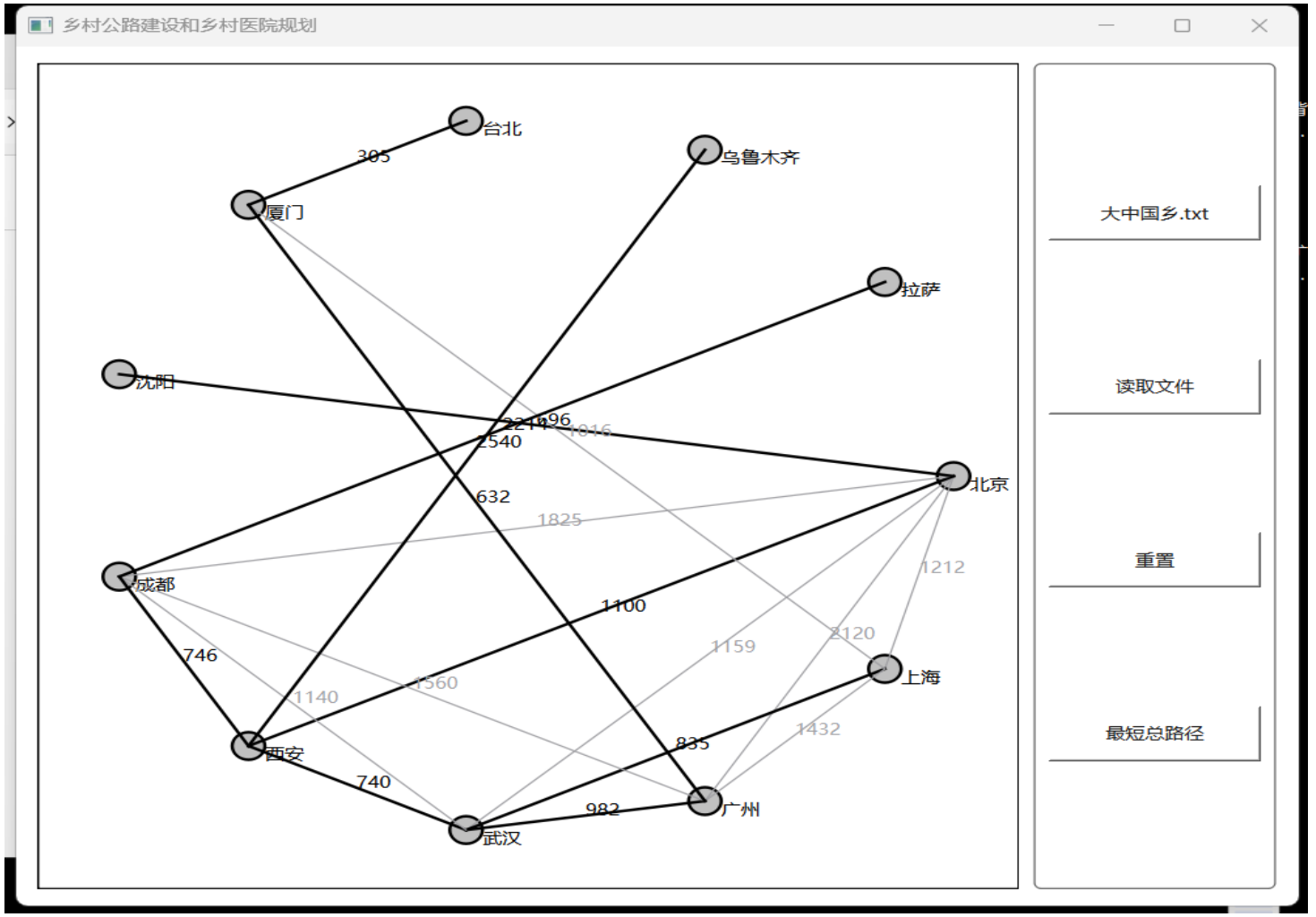
读取文件功能展示

02


乡村公路建设和乡村医院规划
台北
乌鲁木齐
厦门
拉萨
沈阳
北京
成都
上海
西安
广州
武汉
305
2216
1016
2540
632
1825
1212
1100
746
1159
2120
1560
1140
835
1432
740
982
大中国乡.txt
读取文件
最小生成树
最短总路径

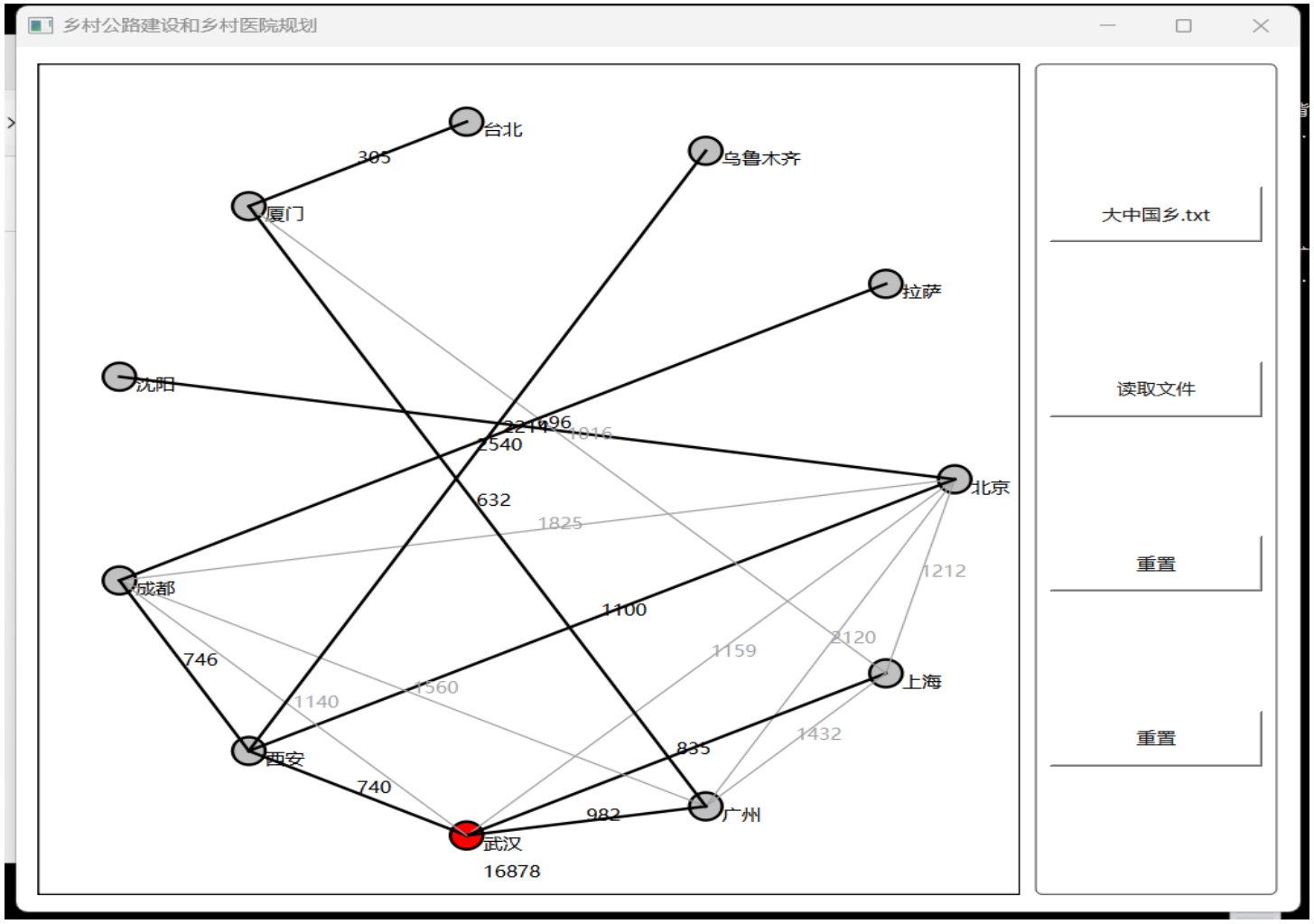
最小生成树功能展示

03



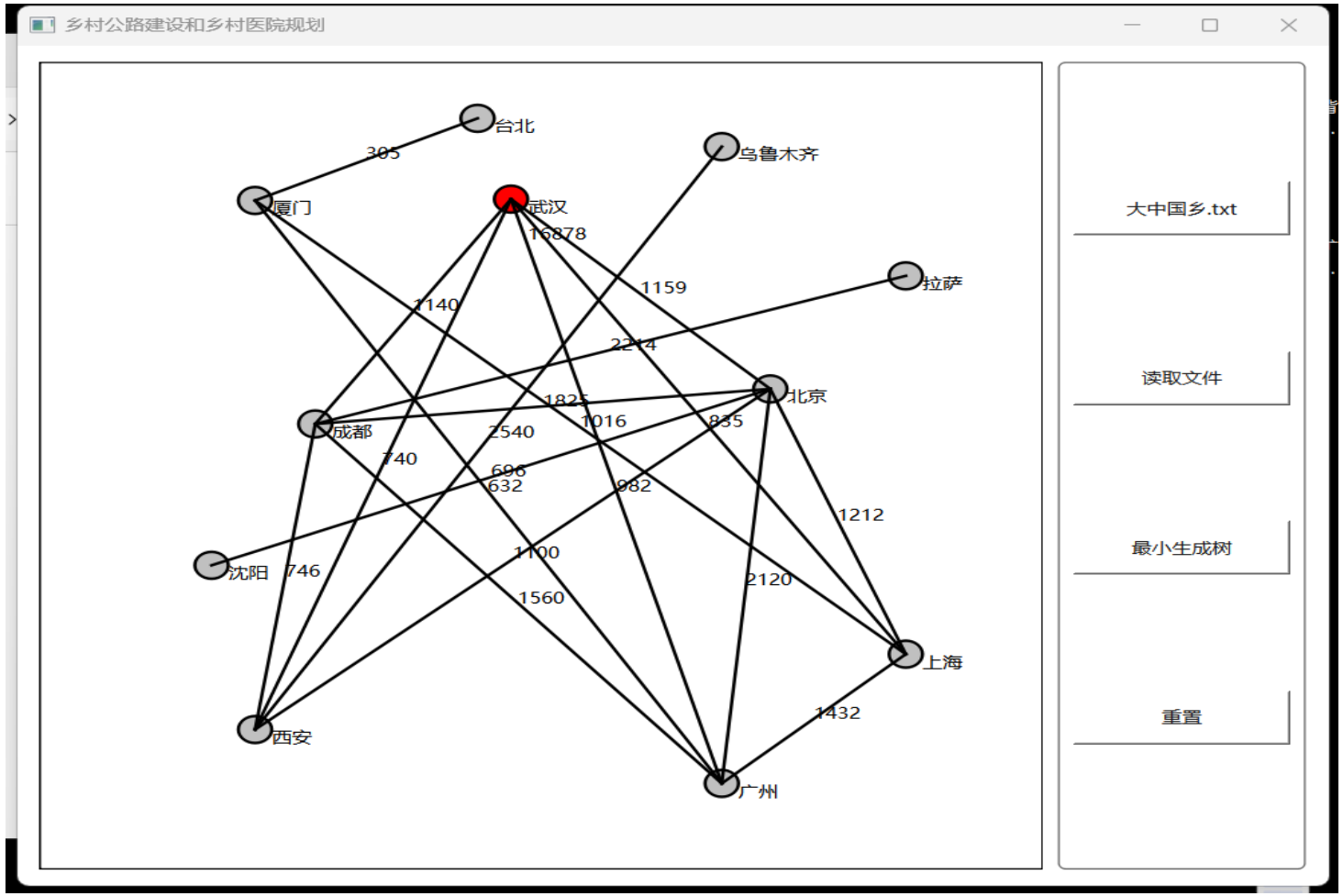
最短总路径功能展示

04



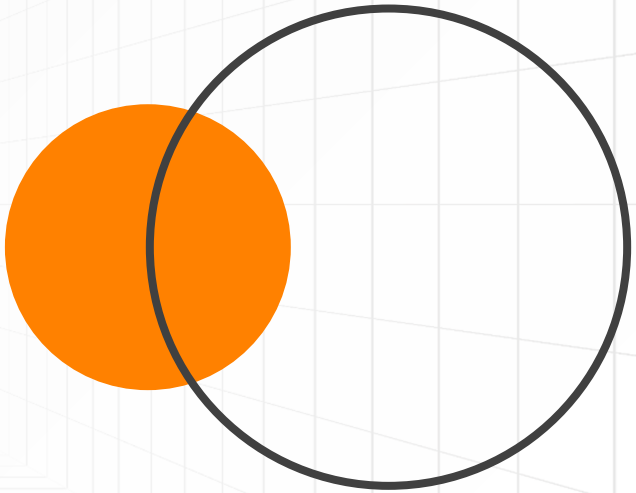
移动图形中各点的位置，可以在新的图形上重复以上操作

05



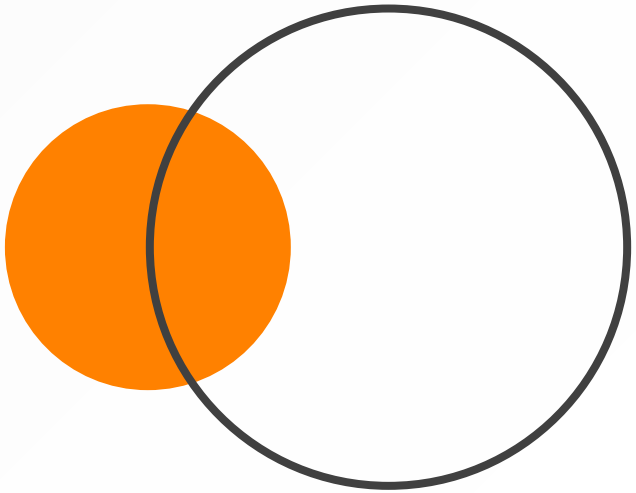
04

章节4 总结与心得



使用了 Qt 框架设计了直观友好的用户界面

本次作业中，采用 Kruskal 算法找最小成本公路网络，用 Floyd 算法算最短总路径，用邻接表表示图结构，算法选择考虑问题特点。展示方面使用 Qt 框架，设计直观友好界面，包括简洁清晰交互、结果可视化、用户反馈。处理 C++中中文字符问题，统一用 UTF-8 编码，使用 std::wstring 存储字符串，并在不同地方用不同转换方式适配。通过作业，小组学习图算法应用与 Qt 框架使用，掌握处理中文字符方法，巩固课程知识，提升团队合作与问题解决能力，是有意义实践活动。



谢谢大家

汇报人小组：第8组