

《软件课程设计》报告

姓名：陶威

学号：U201613331

学院：电子信息与通信学院

专业班级：电信1604班

指导老师：江国星

# 目录

[《软件课程设计》报告 1](#_Toc535273226)

[目录 2](#_Toc535273227)

[Part One：设计说明报告 3](#_Toc535273228)

[需求规定： 3](#_Toc535273229)

[运行环境： 3](#_Toc535273230)

[工程结构： 3](#_Toc535273231)

[程序结构： 4](#_Toc535273232)

[拓扑设计方案说明： 4](#_Toc535273233)

[遍历算法设计说明： 5](#_Toc535273234)

[导航设计算法说明： 6](#_Toc535273235)

[UI绘制程序说明： 7](#_Toc535273236)

[最终程序效果： 9](#_Toc535273237)

[Part Two：测试分析报告 10](#_Toc535273238)

[主要测试内容： 10](#_Toc535273239)

[遍历及导航算法的测试： 10](#_Toc535273240)

[UI绘制程序的测试： 13](#_Toc535273241)

[路径绘制的测试： 13](#_Toc535273242)

[Part Three：开发总结报告 14](#_Toc535273243)

[产品功能说明： 14](#_Toc535273244)

[开发流程： 14](#_Toc535273245)

[程序运行流程： 15](#_Toc535273246)

[开发工作评价： 15](#_Toc535273247)

[经验与教训： 16](#_Toc535273248)

[参考文献： 16](#_Toc535273249)

# Part One：设计说明报告

## 需求规定：

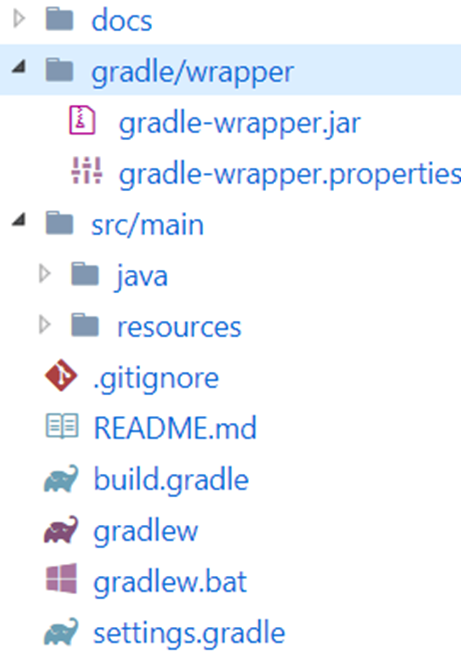
1. 规划最短路线不重复浏览所有人文景点
2. 规划最短路线导航并经过最少一个人文景点
3. 设计一个能表达校园拓扑的方案，并存储在文件中
4. 拓扑图由文件加载，修改文件能增删地点

## 运行环境：

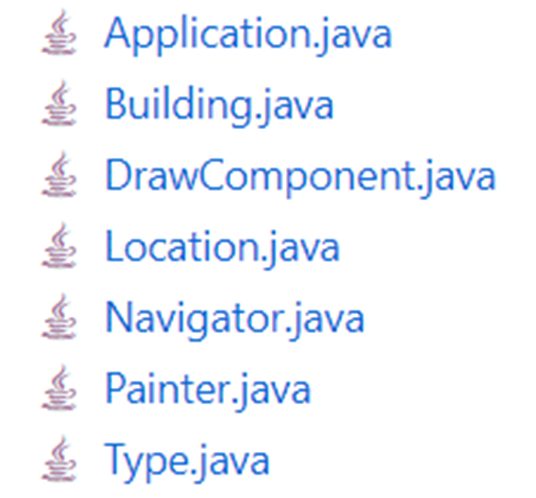
Java 11，macOS Mojave，IntelliJ IDEA

## 工程结构：

使用Gradle构建工具搭建项目骨架，采用约定俗称的目录结构，具体如下：



## 程序结构：



* Application.java：顶层模块，负责程序框架的启动
* Building.java/Location.java：建筑物类/坐标类，存储建筑物的信息（主要是坐标），并附带一些计算距离等方法
* Painter.java/DrawComponent.java：画图能力的抽象，主要负责将JSON信息中的建筑物和导航算法中得到的路径绘制出来
* Navigator.java：导航算法的具体实现，实现了逻辑层面的路径规划和选择
* Type.java：建筑类型的枚举，并实现了字符串转枚举类型的静态方法

## 拓扑设计方案说明：

采用邻接表的数据格式，自定义一套JSON的数据来表示每个坐标位置点的信息，具有可扩展性和可自定义性，详细字段如下：

{

"name": "A", // 建筑名称

"type": "culturalAttraction", // 类型

"longitude": 436, // 经度

"latitude": 224, // 纬度

"availablePlaces": [ // 可到达的地点

"B"

]

}

## 遍历算法设计说明：

* 采用了JSON的自定义坐标格式，因此采用了递归来实现基于“下一跳”的路径遍历
* 主要思想为，遍历所有可到达的且未被遍历的坐标，直至到达起点且所有人文景点被遍历，求出这些路径中的最短距离，即得到最短路径

具体实现如下：

/\*\*

\* 递归遍历所有可达到的人文景点

\* @param origin 起点

\* @param building 当前所处位置

\* @param route 走过的路径

\* @param res 所有结果

\*/

public void travel(String origin, Building building, String[] route, ArrayList<String[]> res) {

for (String availablePlaceName: building.getAvailablePlaces()) {

Building availablePlace = this.getBuildingByName(availablePlaceName);

boolean flag = false;

// 未被遍历过

if(!Arrays.asList(route).contains(availablePlaceName)) {

String[] newRoute = Arrays.copyOf(route, route.length + 1);

newRoute[route.length] = availablePlaceName;

this.travel(origin, availablePlace, newRoute, res);

flag = true;

}

// 到达终点且所有人文景点被遍历则保存路径

if (!flag && availablePlaceName.equals(origin) && this.judgeWhetherAllTraveled(route)) {

String[] newRoute = Arrays.copyOf(route, route.length + 1);

newRoute[route.length] = origin;

res.add(newRoute);

}

}

}

## 导航设计算法说明：

* Dijkstra最短路径算法
* 动态规划，一步步逼近目标
* 具体思想为，使用HashMap保存起点到所有点的距离，遍历可到达的点并更新距离值，遍历过程中如距离未比已存的距离短则直接丢弃该条路径

具体实现如下：

public void findPath(Building start, String end, HashMap distances, String[] route, ArrayList<String[]> res) {

double currentDistance = (double) distances.get(start.getName());

for (String buildingName: start.getAvailablePlaces()) {

if (Arrays.asList(route).contains(buildingName)) {

continue;

}

Building building = this.getBuildingByName(buildingName);

double nextDistance = currentDistance + start.getDistance(building);

double distance = (double) distances.getOrDefault(buildingName, Double.MAX\_VALUE);

if(nextDistance < distance) {

String[] newRoute = Arrays.copyOf(route, route.length + 1);

newRoute[route.length] = buildingName;

distances.put(buildingName, nextDistance);

// 到终点即终止递归

if(buildingName.equals(end)) {

res.add(newRoute);

} else {

this.findPath(building, end, distances, newRoute, res);

}

}

}

}

## UI绘制程序说明：

* 主程序继承自JFame，负责设计界面大小、标题等
* 在一个JPanel里绘制所有按钮，并设置其布局为BorderLayout.NORTH，置于程序顶部
* 在界面中间使用抽象的Painter.java程序，绘制根据Navigator.java计算得到的结果来绘制各种建筑物和路径

具体实现如下：

public static void main(String[] args) {

EventQueue.invokeLater(() -> {

JFrame frame = new Application();

Image icon = new ImageIcon("./src/main/resources/icons/navigator.png").getImage();

frame.setIconImage(icon);

frame.setSize(800, 600);

frame.setTitle("校园导航系统");

frame.setLocationRelativeTo(null);

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setVisible(true);

});

}

public Application() {

this.painter = new Painter(this.navigator);

this.painter.addMouseListener(new MouseHandle());

this.add(this.painter);

this.drawButton(this.navigator.getAllBuildingsName());

this.pack();

}

public JComboBox setComboBox(JPanel panel, String[] options, String label) {

JLabel endLabel = new JLabel(label);

panel.add(endLabel);

JComboBox comboBox = new JComboBox();

for(String option: options) {

comboBox.addItem(option);

}

panel.add(comboBox);

return comboBox;

}

public void drawButton(String[] buildingsName) {

JPanel contentPane = new JPanel();

this.startComboBox = this.setComboBox(contentPane, buildingsName, "选择起点：");

contentPane.add(this.startComboBox);

JButton travelButton = new JButton("游览人文景点");

travelButton.addActionListener(event -> {

this.painter.setStart((String) this.startComboBox.getItemAt(this.startComboBox.getSelectedIndex()));

this.painter.setMode(Painter.TRAVEL\_MODE);

});

contentPane.add(travelButton);

this.endComboBox = this.setComboBox(contentPane, buildingsName, "选择终点：");

contentPane.add(this.endComboBox);

JButton navigateButton = new JButton("开始导航");

navigateButton.addActionListener(event -> {

this.painter.setStart((String) this.startComboBox.getItemAt(this.startComboBox.getSelectedIndex()));

this.painter.setEnd((String) this.endComboBox.getItemAt(this.endComboBox.getSelectedIndex()));

this.painter.setMode(Painter.NAVIGATOR\_MODE);

});

contentPane.add(navigateButton);

this.add(contentPane, BorderLayout.NORTH);

}

public void drawAllBuildingsAndRoads(Graphics g, Building[] buildings) {

for(Building building: buildings) {

this.drawBuilding(g, building);

for(String availablePlaceName: building.getAvailablePlaces()) {

Building availablePlace = this.navigator.getBuildingByName(availablePlaceName);

if (availablePlace == null) {

continue;

}

float x1 = building.getLongitude();

float y1 = building.getLatitude();

float x2 = availablePlace.getLongitude();

float y2 = availablePlace.getLatitude();

this.drawLine((Graphics2D) g, x1, y1, x2, y2, new Color(120, 165, 240));

g.setColor(new Color(61, 77, 102));

g.drawString(Long.toString(Math.round(building.getDistance(availablePlace))), (int) (x1 + x2) / 2 + 10, (int) (y1 + y2) / 2 - 10);

}

}

}

public void drawPath(Graphics2D g2, String[] path) {

for (int i = 0; i < path.length - 1; i++) {

Building building = this.navigator.getBuildingByName(path[i]);

Building nextBuilding = this.navigator.getBuildingByName(path[i + 1]);

float x1 = building.getLongitude();

float y1 = building.getLatitude();

float x2 = nextBuilding.getLongitude();

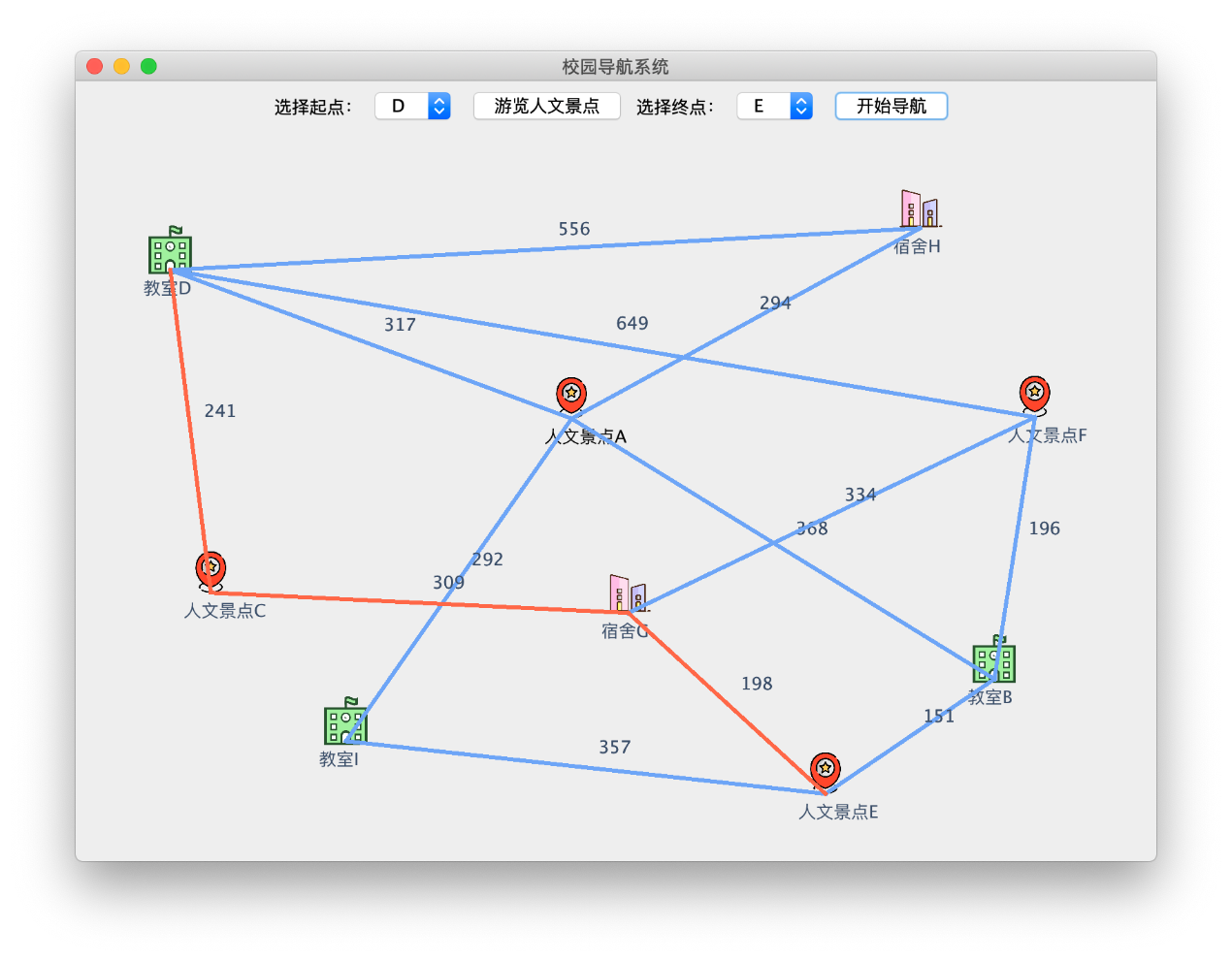
float y2 = nextBuilding.getLatitude();

this.drawLine(g2, x1, y1, x2, y2, new Color(251, 114, 83));

}

}

## 最终程序效果：



# Part Two：测试分析报告

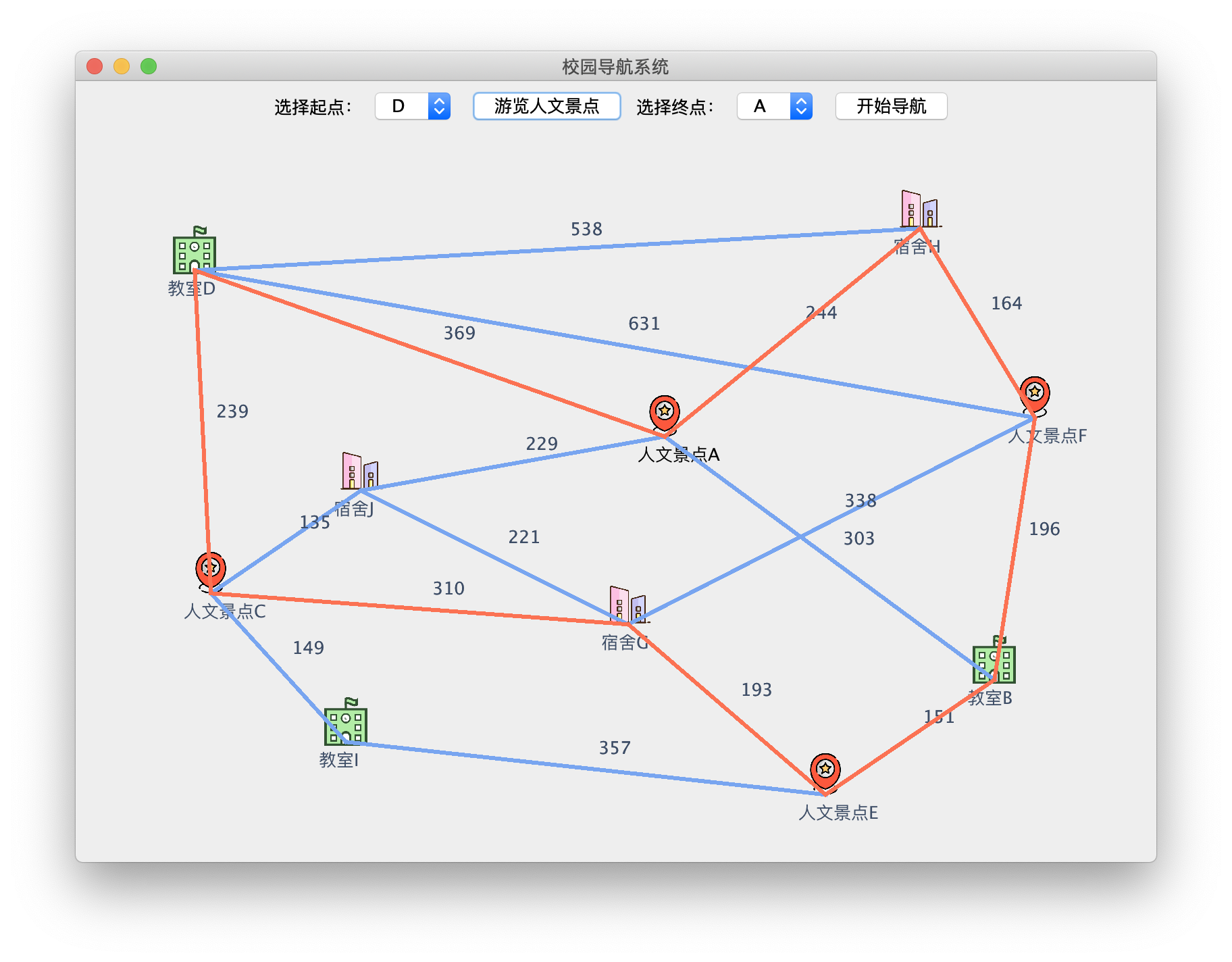
## 主要测试内容：

* 遍历和导航算法的正确与否
* 建筑物节点及可达到路径绘制是否准确
* 根据计算结果绘制的遍历/导航路径是否正确

## 遍历及导航算法的测试：

1. 设计一份拓扑结构图，可以逐渐从简单到复杂
2. 在遍历和导航算法返回结果之前，遍历结果路径并打印每一步节点名称
3. 对比结果是否符合要求并且可行

**拓扑图结构如下：**

****

**测试遍历算法：**

打印出的所有可实现路径为：

[D, H, F, G, J, C, I, E, B, A, D]

[D, H, F, G, J, A, B, E, I, C, D]

[D, H, F, G, E, I, C, J, A, D]

[D, H, F, G, E, B, A, J, C, D]

[D, H, F, G, C, I, E, B, A, D]

[D, H, F, B, E, G, C, J, A, D]

[D, H, F, B, E, I, C, J, A, D]

[D, H, F, B, E, I, C, G, J, A, D]

[D, H, F, B, A, J, G, E, I, C, D]

[D, H, A, J, G, F, B, E, I, C, D]

[D, H, A, J, G, C, I, E, B, F, D]

[D, H, A, J, C, I, E, G, F, D]

[D, H, A, J, C, I, E, B, F, D]

[D, H, A, J, C, G, E, B, F, D]

[D, H, A, B, F, G, E, I, C, D]

[D, H, A, B, E, I, C, J, G, F, D]

[D, H, A, B, E, I, C, G, F, D]

[D, F, H, A, J, G, E, I, C, D]

[D, F, H, A, B, E, G, J, C, D]

[D, F, H, A, B, E, G, C, D]

[D, F, H, A, B, E, I, C, D]

[D, F, G, J, C, I, E, B, A, H, D]

[D, F, G, J, C, I, E, B, A, D]

[D, F, G, J, A, B, E, I, C, D]

[D, F, G, E, I, C, J, A, H, D]

[D, F, G, E, I, C, J, A, D]

[D, F, G, E, B, A, J, C, D]

[D, F, G, C, I, E, B, A, H, D]

[D, F, G, C, I, E, B, A, D]

[D, F, B, E, G, C, J, A, H, D]

[D, F, B, E, G, C, J, A, D]

[D, F, B, E, I, C, J, A, H, D]

[D, F, B, E, I, C, J, A, D]

[D, F, B, E, I, C, G, J, A, H, D]

[D, F, B, E, I, C, G, J, A, D]

[D, F, B, A, J, G, E, I, C, D]

[D, C, J, G, E, B, F, H, A, D]

[D, C, J, G, E, B, A, H, F, D]

[D, C, J, A, B, E, G, F, H, D]

[D, C, J, A, B, E, G, F, D]

[D, C, I, E, G, J, A, H, F, D]

[D, C, I, E, G, J, A, B, F, H, D]

[D, C, I, E, G, J, A, B, F, D]

[D, C, I, E, G, F, H, A, D]

[D, C, I, E, G, F, B, A, H, D]

[D, C, I, E, G, F, B, A, D]

[D, C, I, E, B, F, H, A, D]

[D, C, I, E, B, F, G, J, A, H, D]

[D, C, I, E, B, F, G, J, A, D]

[D, C, I, E, B, A, J, G, F, H, D]

[D, C, I, E, B, A, J, G, F, D]

[D, C, I, E, B, A, H, F, D]

[D, C, G, E, B, F, H, A, D]

[D, C, G, E, B, A, H, F, D]

[D, A, J, G, F, B, E, I, C, D]

[D, A, J, G, C, I, E, B, F, H, D]

[D, A, J, G, C, I, E, B, F, D]

[D, A, J, C, I, E, G, F, H, D]

[D, A, J, C, I, E, G, F, D]

[D, A, J, C, I, E, B, F, H, D]

[D, A, J, C, I, E, B, F, D]

[D, A, J, C, G, E, B, F, H, D]

[D, A, J, C, G, E, B, F, D]

[D, A, H, F, G, E, I, C, D]

[D, A, H, F, B, E, G, J, C, D]

[D, A, H, F, B, E, G, C, D]

[D, A, H, F, B, E, I, C, D]

[D, A, B, F, G, E, I, C, D]

[D, A, B, E, I, C, J, G, F, H, D]

[D, A, B, E, I, C, J, G, F, D]

[D, A, B, E, I, C, G, F, H, D]

[D, A, B, E, I, C, G, F, D]

计算得到的最短路径为： [D, A, H, F, B, E, G, C, D]，与结果一致

**测试导航算法：**

设置起点为D，终点为G，打印出来的路径如下：

[D, C, G]

与结果一致

## UI绘制程序的测试：

根据JSON文件里定义的坐标对比程序绘制出的UI里建筑物的名称，位置，类型和可达到节点是否一致，图略。

## 路径绘制的测试：

根据*遍历及导航算法的测试*里打印的路径结构来对比绘制出的UI里路径是否正确，图略。

# Part Three：开发总结报告

## 产品功能说明：

* 能够按照JSON文件绘制拓扑图，并且能根据建筑物类型自动显示相应的类型图标
* 能够实现遍历和导航算法，通过下拉框选择起点和终点，点击按钮开始导航
* 能够采用点击方式设置起点和终点：在拓扑图上的建筑物附近，鼠标左键点击可将此节点设置为起点，鼠标右键点击可将此节点设置为终点
* 修改JSON文件后可改变拓扑图结构

## 开发流程：

* **明确需求**：研读题目，确定整体思路和程序框架
* **搭建工程**：使用Gradle工程模板，写好项目骨架
* **规范开发**：分层级，分阶段开发，使用git版本控制工具，规范git commit message
* 分层级，分阶段开发，使用git版本控制工具，规范git commit message
* **运行调试**：测试是否有bug或者不够完善的地方

## 程序运行流程：



## 开发工作评价：

生产效率上，前期进度不是很紧凑，主要在构思和设计，并且同时学习一些可能要用到的技术等；中期主要开始编码工作，写起来比较顺利，到也不是考虑很周全，有时候写到某一部分后又要去改前面的设计；后期主要是测试和改进，通过多次测试找到一些bug，然后去理解算法思路并改进。

产品质量上，感觉功能并不是很人性化，比如增删节点这块没有去设计一个程序来直接更改，没有想远一点，老师第一次验收时要求的可点击也是后来才加上的。不过题目文档里要求的基础功能应该都是没有问题的。

## 经验与教训：

首先是要学会发掘需求，从更长远的角度看待问题，并且程序要留一定的可拓展性，尽量不要把某些功能写死。

其次是要把握好进度，尽量早点动工，可以边学边做，以便早点来测试和完善。

最后是要加强软件工程意识，不能上来就直接写实现，应该先设计好功能和结构，再来分步分块实现。

## 参考文献：

Java核心技术·卷 I