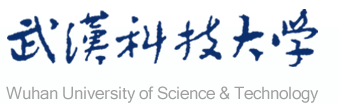
****

《数据结构》实验报告

**专 业：** 计算机科学与技术（卓越计划）

**班 级：** 计算机（卓越）1901班 教学班1454

**学 号：** 201913137031

**姓 名：** 杨尔行

**实验4 图**

**一、【实验目的】**

1. 掌握图的定义和图的邻接表存储结构；
2. 掌握图的创建方法；
3. 掌握顶点和边的操作；
4. 掌握图的基本算法并能实际应用；
5. 掌握图的深度优先搜索算法以及实现方法；
6. 使用C/C++语言和图实现“公交线路图”专题。

**二、【实验环境】**

硬件：Intel Core i5 8300H CPU 2.30GHZ

软件：Windows 10 64bit操作系统，编译环境 Visual Studio 2019

**三、【实验内容】**

程序为控制台程序,使用图数据结构和算法,开始运行时,输出菜单,供用户选择。具体实现的功能如下：  
1）创建公交线路图输入站点和公交线路数据,程序根据站点信息和线路信息创建公交线路图。  
(1)站点信息集合(编号、名字)。  
(2)公交线路信息集合(线路编号,线路两端站点编号,路段长度)。  
2）查询公交线路和站点信息  
为验证公交线路图是否创建成功,程序需实现查询公交线路和站点信息功能。

(1)查询公交线路:输入公交线路编号,系统通过公交线路编号查找到该线路途经的所有站点并输出。

(2)查询站点信息:输人站点编号,系统通过站点编号查找到所有经过该站点的公交线路并输出。  
3)查询两站点之间的路线,找到至多换乘 1 次的路线,并输出结果  
用户输人要查询的起点和终点,程序先判断两个站点之间是否有一条路径（即两个站点之间是否连通)。若两个站点之间有路线,则找到所有最多换乘 1 次的路线,然后依次输出,如图 1-39所示。共找到N条路线,如图1-40所示。  
(1)提示共找到几条路线:从|起点站名|到|终点站名|共找到N条路线,如图1-40所示。

(2)循环依次输出每条路线,有路线编号和站点与公交信息。依次输出路线中经过的每一站,并在站点与站点之间输出两站之间所坐的公交车名,如图 1-41所示。

(3)若两个站点之间没有可以找到的路线,则提示用户“两站点之间没有公交路线!”,如图 1-42所示。若两个站点之间有路线,但是不满足最多换乘 1.次的条件,则提示用户“没有满足条件的路线!”,如图 1-43 所示。

**四、【实验程序】**

1)程序源代码

**文件名：main.cpp**

#include "dialog.h"

int main()

{

    Dialog(R"(D:\Projects\AlgorithmLab\e4\test\_data.txt)")

        .begin\_dialog();

}

**文件名：dialog.h**

#ifndef \_DIALOG\_H\_

#define \_DIALOG\_H\_

#include "map.h"

/// 负责UI层的互动

class Dialog

{

Map map;

/// 查询公交线路

void query\_bus();

/// 查询两点间的全部路径

void query\_path();

/// 查询经过车站的公交线路

void query\_stop();

public:

Dialog(const char\* file\_name);

Dialog(const Map& map);

void begin\_dialog();

};

#endif // \_DIALOG\_H\_

**文件名：map\_scanner.h**

#ifndef \_MAP\_SCANNER\_H\_

#define \_MAP\_SCANNER\_H\_

#include <iostream>

#include "map.h"

/// 从文件流读取地图

class MapScanner

{

public:

    Map scan(std::istream& s);

};

#endif // \_MAP\_SCANNER\_H\_

**文件名：map.h**

#ifndef \_ADJACENCY\_LIST\_H\_

#define \_ADJACENCY\_LIST\_H\_

#include <list>

#include <vector>

#include <string>

/// 顶点索引

using VertexIndex = int;

/// 公交线路索引

using BusIndex = int;

/// 边的额外数据

struct EdgeAddtionalData

{

    /// 长度

    double length = -1;

    /// 属于哪个公交

    BusIndex bus\_index = -1;

    EdgeAddtionalData() {};

    EdgeAddtionalData(double length, BusIndex bus\_index) :length(length), bus\_index(bus\_index) {}

};

/// 顶点的额外数据

struct VertexAddtionalData

{

    /// 顶点（车站）的名字

    std::string name;

    VertexAddtionalData() {}

    VertexAddtionalData(const std::string& name) :name(name) {}

};

/// 边

struct Edge

{

    /// 尾点的索引

    VertexIndex tail;

    /// 额外数据

    EdgeAddtionalData data;

    Edge(VertexIndex tail, EdgeAddtionalData data) : tail(tail), data(data) {}

};

/// 顶点

struct Vertex

{

    /// 所有以它为起始点的边

    std::list<Edge> edges;

    VertexAddtionalData data;

    Vertex() {}

    Vertex(const std::string& name) :data(VertexAddtionalData(name)) {}

    Vertex(const VertexAddtionalData& data) :data(data) {}

};

/// 公交线路信息

struct BusInfo

{

    /// 公交车名称

    std::string name;

    /// 公交车起始站

    VertexIndex start;

    BusInfo() {}

    BusInfo(const std::string& name, VertexIndex start) :name(name), start(start) {}

};

/// 地图

class Map

{

    /// 邻接链表

    std::vector<Vertex> vertex\_adjacency;

    /// 所有公交线路

    std::vector<BusInfo> buses;

    /// 深度优先搜索

    void dfs(VertexIndex src, VertexIndex dest

        , std::vector<bool>& visited

        , std::vector<Edge>& local\_path

        , std::vector<std::vector<Edge>>& res);

public:

    /// 添加顶点

    VertexIndex add\_vertex(std::string name);

    /// 添加边

    void add\_edge(VertexIndex vstart, VertexIndex vend, EdgeAddtionalData edge\_data);

    /// 添加公交线

    BusIndex add\_bus(BusInfo info);

    /// 根据名字找公交线

    BusIndex find\_bus(const std::string& bus\_name);

    /// 根据名字找顶点（公交站）

    VertexIndex find\_vertex(const std::string& name);

    /// 根据索引取得公交信息

    BusInfo get\_bus\_info(BusIndex i);

    /// 根据索引取得顶点（公交站）信息

    VertexAddtionalData get\_vertex\_data(VertexIndex i);

    /// 沿着公交线找到所有顶点

    std::vector<VertexIndex> track\_down\_bus(BusIndex bus);

    /// 找两点之间的所有路径

    std::vector<std::vector<Edge>> search\_paths(VertexIndex a, VertexIndex b);

    /// 尾为该顶点的边

    std::vector<Edge> inbound\_edges(VertexIndex v);

    /// 头为该顶点的边

    std::vector<Edge> outbound\_edges(VertexIndex v);

};

#endif // \_ADJACENCY\_LIST\_H\_

**文件名：dialog.cpp**

#include "dialog.h"

#include "map\_scanner.h"

#include <iostream>

#include <fstream>

Dialog::Dialog(const char\* file\_name)

{

    std::cout << "Loading map from " << file\_name << std::endl;

    MapScanner scanner;

    map = scanner.scan(std::ifstream(R"(D:\Projects\AlgorithmLab\e4\test\_data.txt)"));

    std::cout << "Map loaded!" << std::endl;

}

Dialog::Dialog(const Map& map) :map(map) {}

void Dialog::begin\_dialog()

{

    int number = 0;

    while (true) {

        std::cout << "Hi, what do you want me to do?" << std::endl;

        std::cout << "1: Show bus route." << std::endl;

        std::cout << "2: Find paths from A to B." << std::endl;

        std::cout << "3: Show bus stop info." << std::endl;

        std::cout << "0: Exit." << std::endl;

        std::cout << "Enter a number:" << std::endl;

        std::cout << ">>> ";

        std::cin >> number;

        if (number == 0) {

            return;

        }

        else if (number == 1) {

            query\_bus();

        }

        else if (number == 2) {

            query\_path();

        }

        else if (number == 3) {

            query\_stop();

        }

        std::cout << std::endl;

    }

}

void Dialog::query\_bus()

{

    std::string bus\_name;

    std::cout << "Which bus? Tell me its name:" << std::endl;

    std::cout << ">>> ";

    std::cin >> bus\_name;

    BusIndex bi = map.find\_bus(bus\_name);

    if (bi != -1) {

        std::vector<VertexIndex> path = map.track\_down\_bus(bi);

        for (int i = 0; i < path.size(); i++) {

            if (i != 0) {

                std::cout << " -> ";

            }

            std::cout << map.get\_vertex\_data(path[i]).name;

        }

        std::cout << std::endl;

    }

    else {

        std::cout << "No such bus." << std::endl;

    }

}

void Dialog::query\_path()

{

    std::string a, b;

    std::cout << "Enter A and B:" << std::endl;

    std::cout << ">>> ";

    std::cin >> a >> b;

    VertexIndex ai, bi;

    if ((ai = map.find\_vertex(a)) != -1 &&

        (bi = map.find\_vertex(b)) != -1) {

        auto paths = map.search\_paths(ai, bi);

        if (paths.size() == 0)

            std::cout << "Not reachable." << std::endl;

        else {

            int pi = 1;

            for (auto&& path : paths) {

                std::cout << "Path " << pi++ << ": " << std::endl;

                std::cout << map.get\_vertex\_data(ai).name << " ";

                for (auto&& e : path) {

                    auto bus\_name = map.get\_bus\_info(e.data.bus\_index).name;

                    auto stop\_name = map.get\_vertex\_data(e.tail).name;

                    std::cout << "  >"

                        << bus\_name << ">  "

                        << stop\_name;

                }

                std::cout << std::endl;

            }

        }

    }

    else {

        std::cout << "No such stops." << std::endl;

    }

}

void Dialog::query\_stop()

{

    std::string stop\_name;

    std::cout << "Which stop? Tell me its name:" << std::endl;

    std::cout << ">>> ";

    std::cin >> stop\_name;

    VertexIndex vi = map.find\_vertex(stop\_name);

    auto inbounds = map.inbound\_edges(vi);

    auto outbounds = map.outbound\_edges(vi);

    std::cout << "Inbound bus routes:";

    for (Edge e : inbounds) {

        auto bus\_name = map.get\_bus\_info(e.data.bus\_index).name;

        std::cout << bus\_name << "  ";

    }

    std::cout << std::endl;

    std::cout << "Outbound bus routes:";

    for (Edge e : outbounds) {

        auto bus\_name = map.get\_bus\_info(e.data.bus\_index).name;

        std::cout << bus\_name << "  ";

    }

    std::cout << std::endl;

}

**文件名：map\_scanner.cpp**

#include "map\_scanner.h"

#include <string>

#include <boost/regex.hpp>

#include "map.h"

// 定义此宏来禁止加载信息的输出

#define SUPPRESS\_LOADING\_MESSAGES

Map MapScanner::scan(std::istream& strm)

{

    using namespace boost;

    Map map;

    smatch matches;

    regex r\_vertex(R"(^vertex{name:([a-zA-Z0-9]+)})");//匹配顶点声明

    regex r\_bus(R"(^bus{name:([a-zA-Z0-9]+),start:([a-zA-Z0-9]+)})");//匹配公交线路声明

    regex r\_edge(R"(^edge{from:([a-zA-Z0-9]+),to:([a-zA-Z0-9]+),len:(\d+),of\_bus:([a-zA-Z0-9]+)})");//匹配边声明

    regex r\_comment("//.\*$");//匹配注释

    int lineno = 1;

    while (!strm.eof()) {

        std::string line;

        std::getline(strm, line);

        if (regex\_match(line, matches, r\_vertex)) {

            map.add\_vertex(matches[1].str()); // 添加顶点

#ifndef SUPPRESS\_LOADING\_MESSAGES

            std::cout << "[" << lineno << "]" <<

                " vertex added: " << matches[1].str() << "\n";

#endif

        }

        else if (regex\_match(line, matches, r\_bus)) {

            VertexIndex st = map.find\_vertex(matches[2].str());

            map.add\_bus(BusInfo(matches[1].str(), st)); // 添加线路

#ifndef SUPPRESS\_LOADING\_MESSAGES

            std::cout << "[" << lineno << "]"

                << " bus added: " << matches[1].str() << "\n";

#endif

        }

        else if (regex\_match(line, matches, r\_edge)) {

            VertexIndex v0, v1;

            BusIndex b;

            std::string from\_stop = matches[1].str()

                , to\_stop = matches[2].str()

                , bus\_name = matches[4].str();

            int length = std::stoi(matches[3].str());

            if ((v0 = map.find\_vertex(from\_stop)) != -1

                && (v1 = map.find\_vertex(to\_stop)) != -1

                && (b = map.find\_bus(bus\_name)) != -1) {

                map.add\_edge(v0, v1, EdgeAddtionalData(length, b)); // 添加边

#ifndef SUPPRESS\_LOADING\_MESSAGES

                std::cout << "[" << lineno << "]" <<

                    " edge added: from " << from\_stop

                    << " to " << to\_stop

                    << " of bus " << bus\_name

                    << " with the length of " << length

                    << "\n";

#endif

            }

            else {

#ifndef SUPPRESS\_LOADING\_MESSAGES

                std::cout << "[" << lineno << "]" <<

                    " edge adding error\n";

#endif

            }

        }

        else if (regex\_match(line, matches, r\_comment)) {

        }

        lineno++;

    }

    return map;

}

**文件名：map.cpp**

#include "map.h"

#include <stack>

#include <cassert>

VertexIndex Map::add\_vertex(std::string name)

{

    // 加一条邻接链表

    vertex\_adjacency.push\_back(Vertex(name));

    return vertex\_adjacency.size() - 1;

}

void Map::add\_edge(VertexIndex vstart, VertexIndex vend, EdgeAddtionalData edge\_data)

{

    // 邻接链表加一个节点

    vertex\_adjacency[vstart].edges.push\_back(Edge(vend, edge\_data));

}

BusIndex Map::add\_bus(BusInfo info)

{

    buses.push\_back(info);

    return buses.size() - 1;

}

BusIndex Map::find\_bus(const std::string& bus\_name)

{

    for (int i = 0; i < buses.size(); i++)

        if (buses[i].name == bus\_name)

            return i;

    return -1;

}

VertexIndex Map::find\_vertex(const std::string& name)

{

    for (int i = 0; i < vertex\_adjacency.size(); i++)

        if (vertex\_adjacency[i].data.name == name)

            return i;

    return -1;

}

BusInfo Map::get\_bus\_info(BusIndex i)

{

    return buses[i];

}

std::vector<VertexIndex> Map::track\_down\_bus(BusIndex bi)

{

    VertexIndex vi = get\_bus\_info(bi).start;

    std::vector<VertexIndex> res{ vi };

    bool found\_next = true; // 是否找到下一个的标志

    while (found\_next) { // 如果找到就继续找

        found\_next = false;

        for (auto e : vertex\_adjacency[vi].edges) {

            if (e.data.bus\_index == bi) {

                found\_next = true;

                vi = e.tail;

                res.push\_back(vi);

                break;

            }

        }

    }

    return res;

}

VertexAddtionalData Map::get\_vertex\_data(VertexIndex i)

{

    return vertex\_adjacency[i].data;

}

void Map::dfs(VertexIndex src, VertexIndex dest

    , std::vector<bool>& visited

    , std::vector<Edge>& local\_path

    , std::vector<std::vector<Edge>>& res)

{

    if (src == dest) {

        res.push\_back(local\_path);// 如果找到，将该路径加入res路径集合

        return;

    }

    visited[src] = true;// visited设为true，防止回环

    for (auto&& e : vertex\_adjacency[src].edges) {

        if (!visited[e.tail]) {

            local\_path.push\_back(e);// 将边加入路径

            dfs(e.tail, dest, visited, local\_path, res);// 递归

            local\_path.pop\_back();// 将边删除

        }

    }

    visited[src] = false; // visited重设为false，防止干扰其他路线

}

std::vector<std::vector<Edge>> Map::search\_paths(VertexIndex a, VertexIndex b)

{

    std::vector<std::vector<Edge>> res;

    std::vector<bool> visited(vertex\_adjacency.size());

    std::vector<Edge> local\_path;

    dfs(a, b, visited, local\_path, res);

    return res;

}

std::vector<Edge> Map::inbound\_edges(VertexIndex v)

{

    std::vector<Edge> res;

    VertexIndex i = 0;

    // 搜索所有顶点即可

    for (VertexIndex i = 0; i < vertex\_adjacency.size();i++) {

        for (auto& e : vertex\_adjacency[i].edges) {

            if (e.tail == v) {

                res.push\_back(e);

            }

        }

    }

    return res;

}

std::vector<Edge> Map::outbound\_edges(VertexIndex v)

{

    std::vector<Edge> res;

    VertexIndex i = 0;

    for (auto& e : vertex\_adjacency[v].edges) {

        res.push\_back(e);

    }

    return res;

}

**文件名：test\_data.txt**

// 本文件在程序启动时加载（见main.cpp和dialog.cpp），用于描述公交线路图

vertex{name:GreenAvenue}

vertex{name:BlueRoad}

vertex{name:RedStreet}

vertex{name:YellowLake}

vertex{name:BlackHill}

bus{name:AquaLine,start:BlueRoad}

bus{name:OrangeLine,start:BlueRoad}

bus{name:LimeLine,start:GreenAvenue}

edge{from:BlueRoad,to:GreenAvenue,len:5,of\_bus:AquaLine}

edge{from:GreenAvenue,to:RedStreet,len:4,of\_bus:AquaLine}

edge{from:RedStreet,to:YellowLake,len:3,of\_bus:AquaLine}

edge{from:YellowLake,to:BlackHill,len:6,of\_bus:AquaLine}

edge{from:BlueRoad,to:RedStreet,len:7,of\_bus:OrangeLine}

edge{from:RedStreet,to:GreenAvenue,len:5,of\_bus:OrangeLine}

edge{from:GreenAvenue,to:YellowLake,len:4,of\_bus:OrangeLine}

edge{from:GreenAvenue,to:BlackHill,len:4,of\_bus:LimeLine}

edge{from:BlackHill,to:YellowLake,len:5,of\_bus:LimeLine}

edge{from:YellowLake,to:RedStreet,len:7,of\_bus:LimeLine}

2)程序运行结果截图

AquaLine线

GreenAvenue

BlueRoad

RedStreet

YellowLake

BlackHill

OrangeLine线

LimeLine线

图1：公交路线地图

图2：邻接表

Green

Avenue

BlackHill

YellowLake

RedStreet

BlueRoad

BlackHill

YellowLake

RedStreet

^

YellowLake

^

BlackHill

RedStreet

^

YellowLake

Green

Avenue

^

Green

Avenue

RedStreet

^

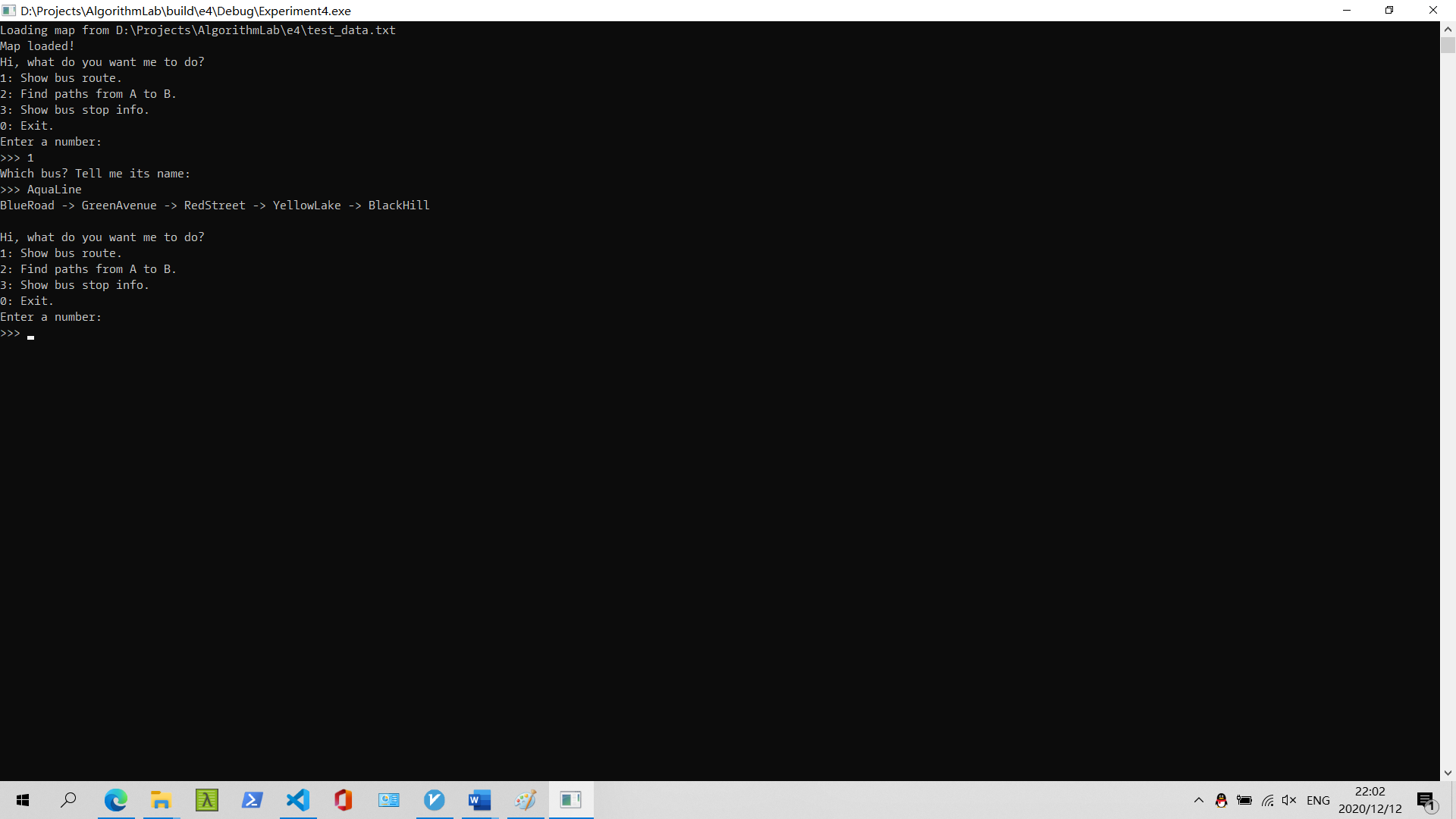


图3：显示某条公交路线

**说明：输入“1”，表示用户要显示某条公交线路**

**说明：输入”**AquaLine”**，表示显示AquaLine公交线（图1中用蓝色表示）**

***说明：输入BlueRoad BlackHill，表示用户要显示从BlueRoad站到BlackHill站的全部路径***

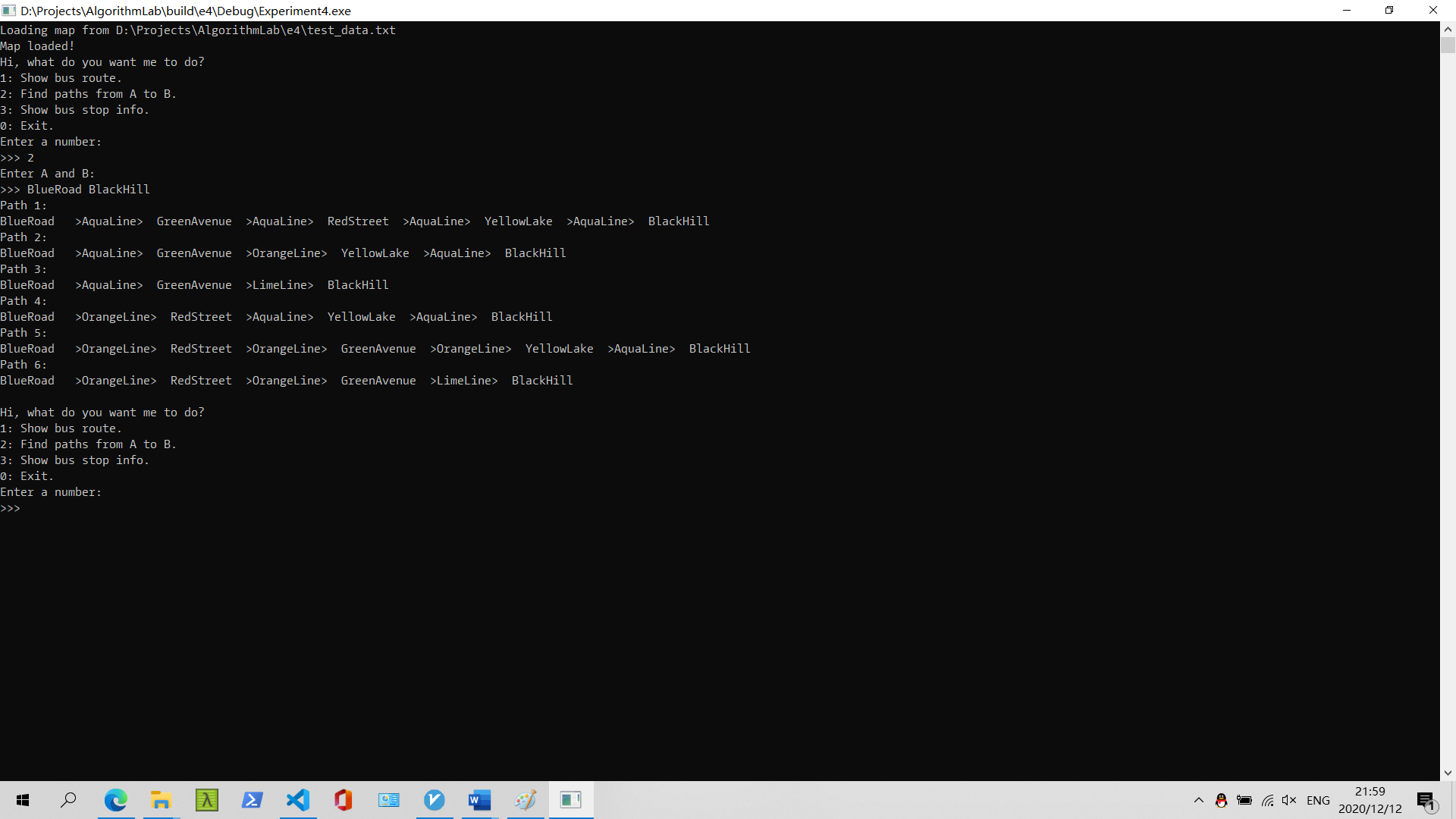
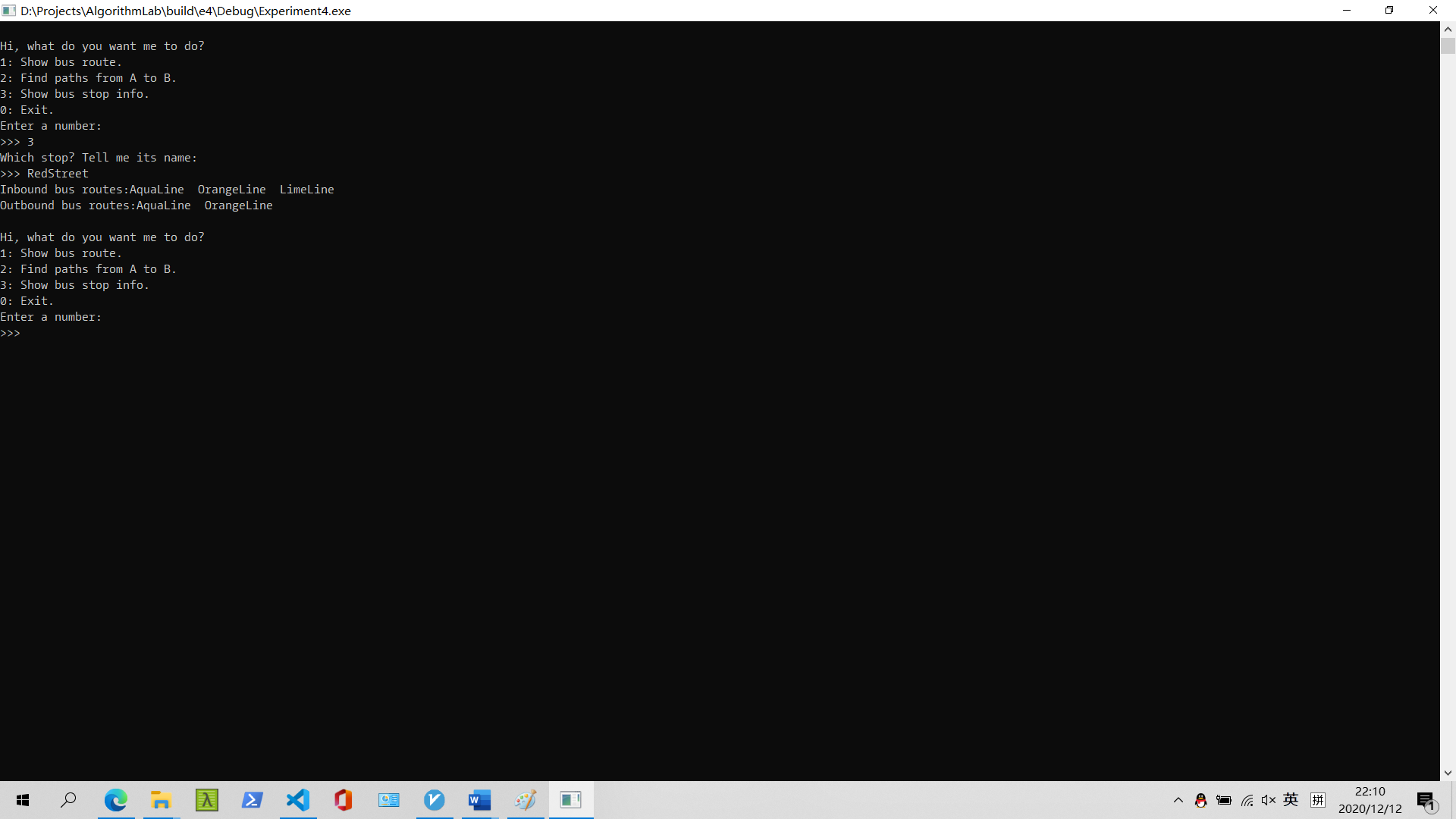


图4：显示两站之间的所有路线

**说明：输入“2”，表示用户要显示两站点之间的全部路径**

**说明：输出”**BlueRoad >AquaLine> GreenAvenue**”表示乘坐AquaLine公交线从BlueRoad到GreenAvenue**

图5：显示某站的路线



***说明：输入***3***，表示用户要显示进出某站的所有公交线***

***说明：输入***RedStreet***表示显示RedStreet站的进出公交***

**五、【实验小结】**

本次实验运用了图的知识。整个公交线路图是一个多重图，本次实验使用了邻接链表来存储。为车站和路段命名、为路段指定长度需要额外为顶点和边绑定数据，但它们不影响算法的执行，所以本次实验我将他们放在了“额外数据”结构体中。

最重要的第二个功能“查找全部路线”使用的是深度优先搜索。考虑到要求搜索全部路径，故对搜索进行一些修改。在维护visited表之外，再增设local\_path表来记录当前节点到根节点的路径（经过的边）。每深入到下一节点，就将此节点的visited设为true，并将边加入local\_path；每回退到上一节点，就将此节点的visited设为false，并将边删除。设置为visited的节点不会被展开。当访问到目的地节点时不退出，而是将local\_path输出。如此，便可以保证找到全部简单路径，且路径之间可以交叉，互不影响。