计算机与软件学院

《程序设计基础 II 课程设计》报告

（2022/2023 学年 第二学期）

课程设计题目 校园导航系统

专　业 软件工程

班　级 22 软工智能 7 班

小组成员 席振华　韩校熠　朱成雨

指导教师 李冬梅

姓名	学号	分工
席振华	2215925792	调试测试，模块架构，数据库，登录注册，DFS 和 BFS
韩校熠	2215925790	算法实现，增改查，结果分析，迪杰斯特拉算法
朱成雨	2215925787	界面设计，删除，逻辑处理，佛洛依德算法

2023 年 6 月

计算机与软件学院

《程序设计基础 II 课程设计》报告

（2022/2023 学年 第二学期）

课程设计题目 校园导航系统

学　号 2215925792

姓　名 席振华

成绩类别	成　绩
课程设计报告（60）	
课程设计答辩（40）	
总成绩（100）	
等级制成绩	

2023 年 6 月

目　　录

校园导航系统

—南阳理工学院地图导航

1. 题目简述

校园导航系统是一个为高校学生和教职员工设计的手机应用程序，旨在提供可靠和准确的校园导航服务，帮助用户方便、快速地找到目的地。该系统将采用定位技术和地图导航技术，结合学校内部的地理信息系统，实现室内外校园导航，包括建筑物、教学楼、宿舍楼、实验室等各种地点，并提供周边服务信息检索、公共交通线路查询等功能。

设计目标包括以下方面：1.室内外定位导航：通过定位技术和地图导航技术，结合学校内部的地理信息系统，提供室内外准确的导航服务。定位系统需要高准确度，尽量避免误差和漂移。导航系统需要智能规划路线，根据用户的起点和目的地，为用户提供最优的路径推荐，优化用户的体验。2.校园地图信息：提供详细、全面的校园地图信息，包括分类的建筑物、教学楼、宿舍楼、运动场所等。用户可以在地图上看到位置、标记和详细信息，并通过搜索快速找到目标地点。3.搜索功能：支持用户搜索校园内各种场所和设施。例如，用户可以搜索学生食堂、超市、书店、医务室等等，找到目标位置，提高效率，减少寻找时间。4.界面友好易用：设计界面要求直观、简洁，减少用户使用难度。操作要方便、直接，使用户能够快速学习和使用核心功能。5.实时更新和数据可视化：及时更新校园建筑物和道路的变化，确保数据是准确的和实用的。同时，对用户使用数据进行可视化分析，了解用户的使用习惯和需求，不断优化系统功能设计。6.安全与隐私：系统需要确保数据安全，并提供隐私保护。例如，用户的位置数据不应向外界发布，只能在必要时与特定机构共享。

2. 问题分析

2.1 需求描述

(1)功能需求：

定位和导航功能：用户可以精确定位和导航服务，根据起点和目的地，方便快速地找到目的地，免去迷路和找不到地方的烦恼。

校园地图功能：提供全面、准确、及时的校园地图，详细标记和介绍校园内各种设施和场所，方便用户了解校园布局和设施分布。

周边服务查询：支持用户在学校内部查询周边的服务设施，如食堂、超市、书店、医务室等，以及属于校园的公共交通线路，方便用户实现全面查询周边服务和设施。

(2)非功能需求：

美观性：一个美观的导航系统能够吸引用户的注意力并提升用户体验。使用清晰的图标、直观的导航流程和视觉上的吸引力。

实用性：一个实用的导航系统应该能够提供准确和及时的信息，帮助用户快速找到他们需要的地点。系统应该具备搜索功能、路线规划功能、地点标记功能等，以满足不同用户的需求。

可扩展性：随着校园的发展和变化，导航系统可能需要添加新的地点、更新地图数据或增加功能。一个可扩展的导航系统应该具备灵活的架构和良好的系统设计，以方便系统的增加和修改，同时保持系统的稳定性和性能。

技术支持：技术支持可以涉及系统的故障排查、技术指导、系统更新和安全性保障等方面。

2.2 系统功能描述

(1)登录、注册、注销、修改密码(此功能仅对管理员开放)；

(2)管理员操作界面的功能实现:增加、删除、修改等功能；

(3)核心功能实现：打印学校平面图，显示学校所有可达的路径及权值，迪杰斯特拉和佛洛依德算法实现最短路径及单源顶点到其他所有路径的算法，DFS 和 BFS 分别实现两顶点间的所有路径及附近周围 x 距离内的所有可达地点路径，查看校园某地点的介绍以及位置信息，定制参观校园专线功能等。

3. 数据结构定义

3.1 逻辑结构定义

采用图逻辑结构，图是由一组节点和一组边组成的数据结构。节点表示实体，边表示节点之间的连接关系。特点：多对多关系，有向性与无向性，权重，连通性连通图。图逻辑结构广泛应用于许多领域，如社交网络分析、路线规划、网络拓扑分析等。它可以帮助解决许多复杂的问题，并提供了有效的数据组织和操作方式。

3.2 顺序存储结构及定义

(1)连续的内存空间：顺序存储结构使用一段连续的内存空间，这使得数据元素在内存中存储紧密，相邻元素之间没有额外的空间开销。它具有常数时间复杂度 O(1)的访问时间，即可以直接通过索引访问元素，无需遍历或搜索。

(2)随机访问：由于元素在内存中的位置是固定的，顺序存储结构支持随机访问。通过索引，我们可以立即访问任何一个元素，而不需要遍历整个数据结构。这对于需要频繁访问特定元素的情况非常有效。。

(3)存储效率：线性存储结构通常具有较低的存储空间开销。相比于其他数据结构，如邻接矩阵或邻接表，线性存储结构可以更高效地存储节点和边的数据。这对于大规模校园地图来说尤为重要，因为校园环境通常有许多节点和边需要存储。

4. 算法设计

4.1 描述算法的主要思想

4.1.1 DFS

DFS(深度优先搜索)查找两个顶点之间的所有路径和权值。首先初始化相关数组和变量。然后调用 DFS 函数进行深度优先搜索，查找从顶点 i 到顶点 j 的所有路径和权值。如果最终未找到路径，即标志变量 f 仍为 false，则输出路径不存在的提示信息。

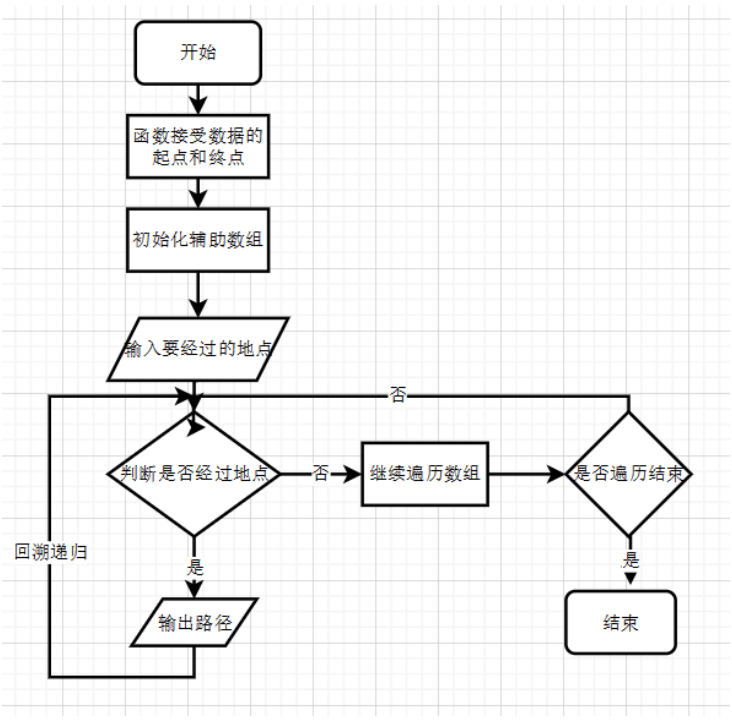

图 4.1 深度优先搜索

4.1.2 BFS

BFS(广度优先搜索)来查找起始顶点周围一定距离(x)范围内的位置路线。首先，代码会初始化一些数组和变量，包括 visited 用于标记顶点是否

已访问，path 用于存储当前顶点的上一个顶点，以及 dist 用于存储每个顶点到起始顶点的距离。接下来，代码创建一个队列 q 用于存储顶点，将起始顶点加入队列中，并将其标记为已访问。起始顶点到自身的距离设置为 0。然后，进入循环，直到队列为空。在每次循环中，取出队头顶点 curr，并遍历与其相邻的顶点。如果某个相邻顶点尚未访问且与当前顶点有连接的边（权值不为无穷大），则更新该相邻顶点到起始顶点的距离和记录上一个顶点。如果距离仍在限制范围内，则将该相邻顶点加入队列并标记为已访问。搜索结束后，代码会输出所有符合条件的路径和距离。对于每个符合条件的顶点 i，输出起始顶点到该顶点的路径和距离。然后，通过回溯从终点到起点，将路径顶点逐个存入栈 stack<int> s 中。最后，将栈中的顶点弹出，按正序输出路径。

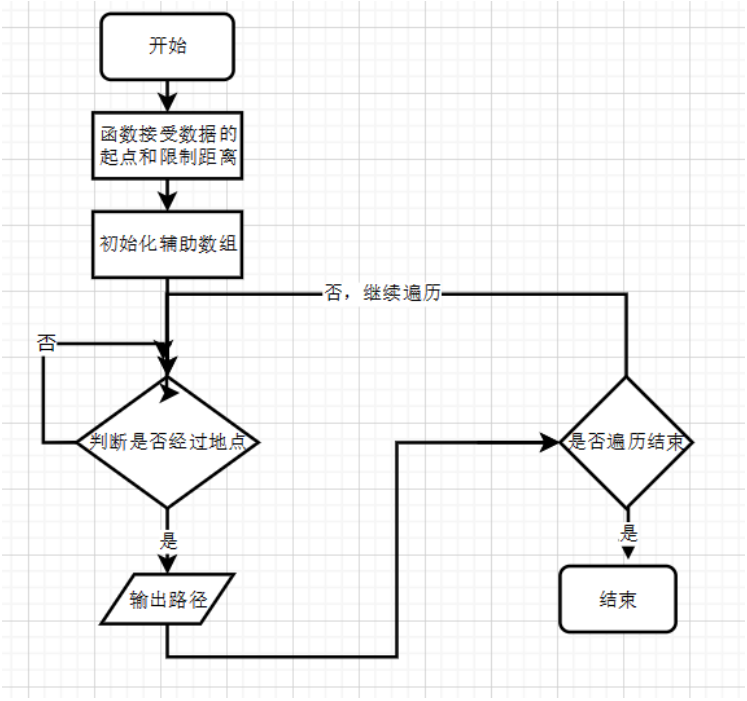

图 4.2 广度优先搜索

4.1.3 Floyed 算法

打印经过指定起点和终点的必经点路径。调用 Floyed() 函数，该函数实现了弗洛伊德算法，用于计算图中所有节点对之间的最短路径。检查从起点到终点的最短路径长度是否为最大值（表示不可达）。如果是，则返回 false，表示无法找到经过指定起点和终点的路径。根据弗洛伊德算法计算的路径矩阵 pathaway，从起点一直向终点迭代，输出经过的节点数据内容，并将节点数据存储到字符串 str 中。输出终点的数据内容和从起点到终点的最短路径长度。

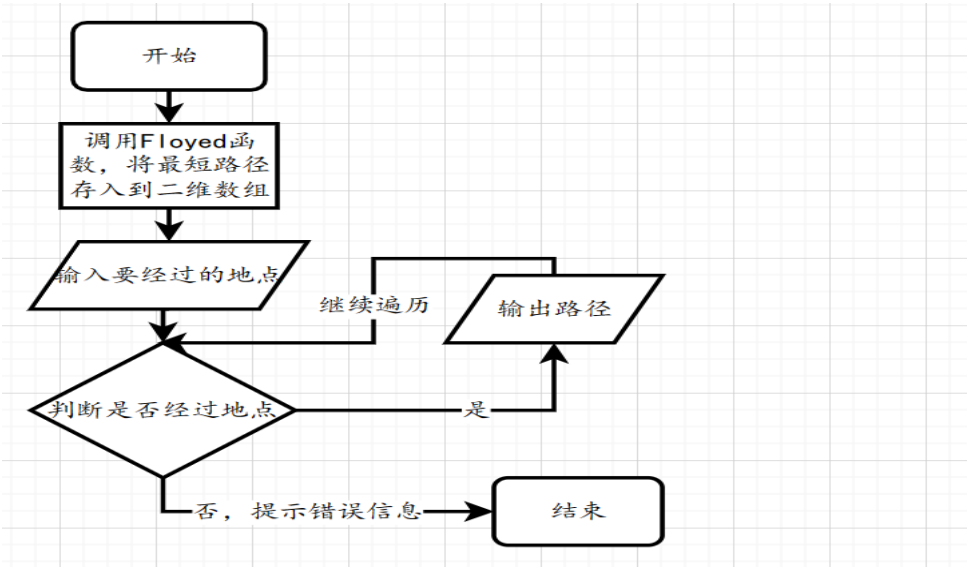

图 4.3 必经点最短路径算法

4.2 算法主要步骤（伪代码分析）

1. DFS(深度优先遍历)

```
procedure DFS(i, j, path, len, sum, f)
    visited[i] ← true
    append i to path
    len ← len + 1
    if i = j then
        f ← true
        for each k from 0 to len - 2 do
            output arrary[path[k]].get_data() + " -> "
        end for
        output arrary[path[len - 1]].get_data() + "  (" + sum +
")" + newline
    end if
    for each k from 0 to vernum - 1 do
        if ArcArrary[i][k] ≠ MAXINFNUM and not visited[k] then
            sum ← sum + ArcArrary[i][k]  // 向深处遍历则增加权值
            DFS(k, j, path, len, sum, f)
            sum ← sum - ArcArrary[i][k]  // 回溯就要减去加上的权
值
        end if
    end for
```

```
    len ← len - 1  // 每次回溯，当 len=0 时也意味着着递归结束
    visited[i] ← false
end procedure
调用 DFS(i, j, empty array of size MAX_SIZE, 0, 0, false)
```

2. BFS(广度优先遍历)

```
queue q//存储顶点的队列
push start into q//将起始顶点加入队列
set visited[start] to true//标记起始顶点已访问过
dist[start] ← 0// 起始顶点到自身的距离为 0
while q is not empty do
    curr ← front element of q  //取出队头顶点
    pop q
    for i from 0 to vernum - 1 do
        if ArcArrary[curr][i] ≠ MAXINFNUM and not visited[i] then
// 当前顶点的相邻顶点未访问过且有边相连
            dist[i] ← dist[curr] + 1  // 更新相邻顶点到起始顶点的
距离
            path[i] ← curr  // 记录上一个顶点
            if dist[i] ≤ limit then  // 如果距离限制内，则将该顶
点加入队列并标记为访问过
                push i into q
                set visited[i] to true
// 输出所有符合条件的路径和距离
for i from 0 to vernum - 1 do
    if visited[i] and dist[i] ≠ 0 and dist[i] ≠ MAXINFNUM then
        output ">路径: " + arrary[start].get_data() + " -> " +
arrary[i].get_data() + "    距离:  " + dist[i] + newline
        curr ← i
        stack s  // 存储路径的栈
        push curr into s
        while curr ≠ start do  // 从终点回溯到起点
            curr ← path[curr]  // 获取上一个顶点
            push curr into s
        end while
```

```
        str ← ""
        while s is not empty do  // 弹出栈中所有顶点，按正序输出路径
            str ← str + arrary[top element of s].get_data() + "->"
            pop s
        end while
        output str.substring(0, str.length() - 2) + newline
    end if
end for
```

3.Floyed 算法(必经点最短路径)

```
function Print_Choice_place(start, end, sum, str)
    Floyed()  // 调用弗洛伊德算法，求最短路径
    if dist_f[start][end] = MAXINFNUM then
        return false
    else
        output arrary[start].get_data() + "->"
        cur ← start
        while pathaway[cur][end] ≠ end do
            cur ← pathaway[cur][end]
            output arrary[cur].get_data() + "->"
            str ← str + arrary[cur].get_data() + "->"
        end while
        output arrary[end].get_data() + "  (" + dist_f[start][end] + ")" + newline
        str ← str + arrary[end].get_data() + "->"
        sum ← sum + dist_f[start][end]  // 累计此路径距离
        return true
    end if
end function
```

4.3 数据库设计

4.3.1 数据库的连接及功能实现

mysql = 初始化 MySQL 对象

如果不成功则：

输出 "连接错误原因" + 获取错误信息(&mysql)

退出程序

如果成功则：

连接到 MySQL 服务器("localhost", "root", "123456", "renai", 0, NULL, 0)

//增加操作

spy = "insert into pathway values('%s','%s',%d);"

sql = 格式化字符串(spy, data1, data2, w)

执行 SQL 语句(sql)

//删除操作

sql = 格式化字符串("delete from place where placemap = '%s';", val)

执行 SQL 语句(sql)

//修改操作

sql1 = 格式化字符串("update pathway set begin = '%s' where begin='%s';", new_na, name)

执行 SQL 语句(sql1)

//查看操作

ssl = "select id from place where placemap='" + data + "';"

执行 SQL 查询(ssl)

result = 获取查询结果()

row = 获取查询结果行(result)

4.3.2 数据库数据表

管理员表里三个数据类型，分别是注册时间、账号、密码，类型分别为 datetime,varchar,varcahr。

time	account	password
2023/5/7 8:4	1	1
2023/6/10 21:7	272682	1
2023/5/7 17:2	880368	1
2023/6/12 13:52	928452	1

图 4.4 管理员表

记录顶点的表里三个数据类型，分别是地点的 id，地点名，地点的简介，类型分别为 int,varchar,varchar。

id	placemap	content
1	北门	南阳理工的大门
2	1教	1
3	3教	1
4	4教	1
5	中关村	1
6	生化学院	1
7	西北操场	1
8	小礼堂	1
9	西北餐厅	1
10	下沉广场	1
11	校医务室	1
12	女生公寓	1
13	土木工程学院	1
14	画室	是发大水发射点
15	建筑学院	1
16	智能制造学院	1
17	齐贤广场	1
18	信息工程学院	1

图 4.5 地点表

记录路径的表里三个数据类型，分别是地点的起点，终点，权值，类型分别为 varchar, varchar, int。

begin	end	path
北门	1教	4
3教	1教	4
中关村	1教	5
1教	生化学院	6
中关村	生化学院	12
西北操场	生化学院	9
4教	生化学院	3
北门	生化学院	7
中关村	小礼堂	2
3教	小礼堂	20
3教	西北餐厅	22
小礼堂	西北操场	17
西北餐厅	下沉广场	1
西北餐厅	女生公寓	1

图 4.6 路径表

5. 算法分析

5.1 DFS(深度优先搜索)

(1)时间复杂度：使用了递归来实现深度优先搜索，递归会对每个未访问的邻接顶点进行遍历。在最坏情况下，每个顶点都需要遍历。因此，时间复杂度可以表示为O(V+E)，其中V是顶点的数量，E是边的数量。在每个递归调用中，需要判断是否已经达到目标顶点，这需要O(V)的时间。因此，在最坏情况下，整个搜索过程的时间复杂度可以表示为O(V*(V+E))。

(2)空间复杂度：使用了一个visited数组和一个path数组来记录访问状态和路径，其大小为顶点的数量，因此空间复杂度为O(V)。在递归调用过程中，还需要考虑函数调用的栈空间。最坏情况下，递归深度可能达到V，因此递归所需的栈空间为O(V)。

(3)总结：从时间和空间复杂度的角度来看，代码的效率可以接受，但在大规模图上可能存在性能问题。因此，我们可以根据实际情况优化代码，并确保代码的可读行和可维护性。

5.2 BFS(广度优先搜索)

(1)时间复杂度：在每个顶点的邻接顶点遍历过程中，需要判断每个相邻顶点是否已经访问过，并更新距离和路径数组。这个过程需要遍历每个顶点和边，所以时间复杂度可以表示为O(V+E)，其中V是顶点的数量，E是边的数量。最坏情况下，所有顶点都需要进行遍历，所以在整个搜索过程中，时间复杂度为O(V*(V+E))。

(2)空间复杂度：使用了一个visited数组、path数组和dist数组来记录访问状态、路径和距离。它们的大小都是顶点的数量，所以空间复杂度为O(V)。代码还使用了一个队列来存储顶点，其空间复杂度与队列中的元素数量有关。在最坏情况下，队列可能包含所有顶点，所以队列的空间复杂度也为O(V)。

(3)总结：从时间和空间复杂度来看，代码的效率可以接受。然而，在大规模图上可能存在性能问题。因此，我们可以根据实际情况优化代码，并确保代码的可读行和可维护性。

5.3 Floyed算法

(1)时间复杂度分析：初始化部分的两个嵌套循环的时间复杂度为 O(vernum^2)。弗洛伊德算法的主循环由三层嵌套循环组成，每个循环的迭代次数是 vernum，因此总的时间复杂度为 O(vernum^3)。

(2)空间复杂度分析：dist_f 数组和 pathaway 数组的大小都为 vernum * vernum，因此所需的空间复杂度为 O(vernum^2)。函数 Map::Print_Choice_place 调用了 Floyed() 函数，并根据最短路径的结果打印输出。其时间复杂度主要由 Floyed() 函数确定，为 O(vernum^3)。空间复杂度取决于传入的参数 sum 和 str 的大小，对于输入的问题规模来说，这些参数的空间占用可以忽略不计。

(3)总结：需要注意的是，以上复杂度分析假设 vernum（节点数目）是问题规模。如果还有其他操作或数据结构涉及到问题规模，则可能对总体复杂度产生影响。

6. 编码实现

```
//Map 类，核心功能类
class Map
{
private:
 Databases database;//定义数据库的对象，连接数据库
public:
 int ArcArrary[99][99] = { 0 };//邻接矩阵
 VertexNode* arrary;//地图的顶点数组
 int vernum;//地点个数
 int arcnum;//边个数

 bool visited[999] = { false };//标记每个顶点是否被访问过
 int dist[999] = { 0 };//存储源点到各点的最短距离
 int path[999] = { 0 };//存储最短路径上当前节点的前一个节点

 int pathaway[99][99] = { 0 };//最短路径上的前驱节点
 int dist_f[99][99] = { 0 };//两个顶点之间的最短距离
public:
 void Read();//读取数据函数
 void Clear();//销毁函数
 int GetDataIndex(string data);//找到此建筑的下标
 bool RepatGetData(string data);//查重
 void Display_schoolmap();//打印平面地图
 void Search_route();//显示所有地点的编号以及地点间的路径距离
 void Dijkstra(int v);//迪杰斯特拉算法,找最短路径
```

```
 void Floyed();//弗洛伊德算法
 void Print_Floyed(int start, int end);//调用佛洛依德
 bool Print_Choice_place(int start, int end, int& sum, string& str);//打印必经点的路径
 void DFS(int i, int j, int path[], int& len, int& sum, bool& f);//深度优先遍历
 void Print_DFS(int i, int j);//打印两点间所有可能存在的路径
 void BFS(int start, int limit);//广度优先遍历
public://管理员操作
 void Add_newVertex();//添加新的顶点
 void Add_newArc();//添加新边
 void Delete_oldVertex(string data);//删除旧的顶点
 bool Delete_oldArc(int i, int j);//删除旧的边
 bool Alter_ver(int i);//修改顶点
 bool Alter_arc(int i, int j);//修改边
 void Writing_information(int i);//描绘信息
public://登录功能
 void registered();//注册
 bool login(string& aCC);//登录
 void modify(string acc);//修改
 void logout(string acc);//注销
};
//数据库类
class Databases
{
private:
 MYSQL mysql;
 MYSQL_RES* result = nullptr;
 MYSQL_ROW row;
public:
 Databases();//连接数据库
 ~Databases();//析构函数
 bool Determine(string sql);//判断操作是否成功
public://地图操作

 void Read_databases_place(VertexNode*& arr,int& vernum);//数据库信息读取地点
 void Read_databases_pathway(int& acrnum,int vernum, VertexNode* arr, int (&ArcArrary)[99][99]);//读取路径信息
 int GetId(int vernum, VertexNode* arr, string name);

 void Add_databases_place(int &eit,string data,string data1);//添加顶点表信息
```

```
 void Add_databases_pathway(string data1,string data2,int w);//添加边边信息
 bool Delete_databases_patheay(string v1, string v2);//删除边
 void Delete_databases_place(string val);//删除顶点
 bool Alter_daatabases_ver(string name, string new_na);//修改顶点，俩表
 bool Alter_daatabases_arc(string d1, string d2,int w);//修改边
 void Writing_inf(string hua, int id);//输入介绍
public://登录界面
 void Regisitered_database_manger(string d1, string d2,string d3);//管理员的登录
 int Login_database(char ac[], char pa[]);//登录
 void Modify_database(string acc,string p);//功能内部的修改密码
 void Logout_database(string acc);//注销账号
};
//顶点类
class VertexNode
{
private:
 int id;
 string data;
 string content;
public:
 VertexNode():id(0),data(""),content("") {}
 void set_data(string ata) {
        this->data = ata;
 }
 string get_data() {
        return data;
 }
 void set_content(string ta) {
        this->content = ta;
 }
 string get_content() {
        return content;
 }
 void set_id(int ik) {
        this->id = ik;
 }
 int get_id() {
        return id;
 }
```

7. 测试与运行记录

(1)DFS(深度优先搜索)查找两个顶点之间的所有路径和权值。如输入起点建筑学院，终点小礼堂，该功能就会输出这两个顶点间的所有路径。

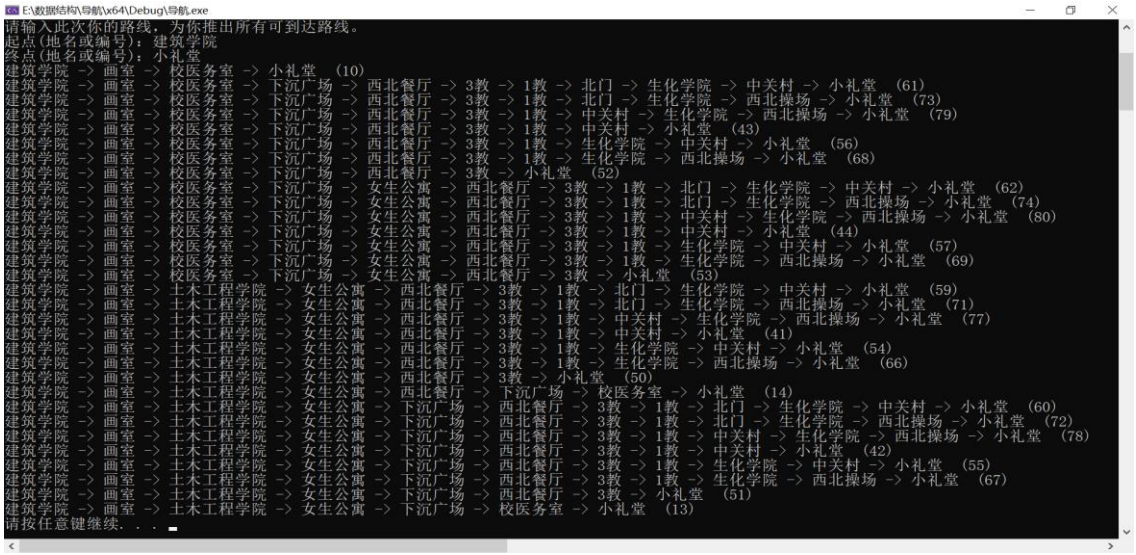

图 7.1 深度优先搜索

(2)BFS(广度优先搜索)来查找起始顶点周围一定距离(x)范围内的位置路线。输入一顶点北门，限制距离为 3，则会输出北门能到达的顶点并且总路径小于 3 的所有路线。

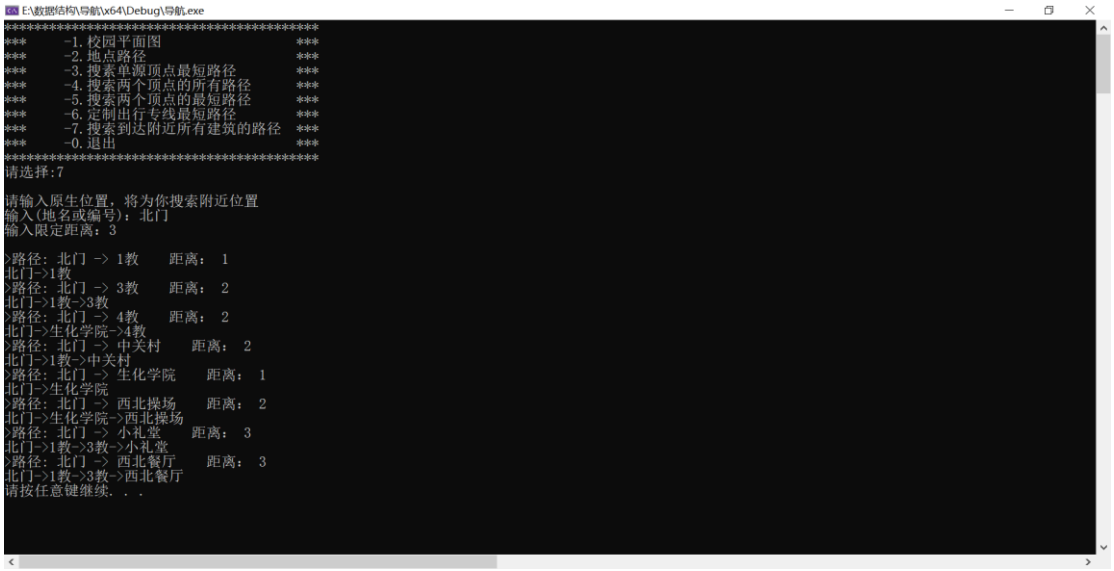

图 7.2 广度优先搜索

(3)必经点最短路径算法。输入起点北门，终点小礼堂和必经点的个数，如示例，输入两个必经点生化学院和体育馆则会输出起点到终点的最短路径并且一定经过这两个顶点。

图 7.3 必经点输出

8. 总结及心得体会

DFS 和 BFS 都是图遍历算法，用于在图中搜索特点节点或路线。以及使用 Floyed 算法完成了定制专线的算法，使代码体验更高级。

在此功能上增加了一些新的功能比如 DFS 实现了输入起点和终点并输出所有有效的路线；以及 BFS 输入一个起点和限制距离，然后根据这两个参数输出起点能到达的所有终点并将路径输出出来。本课题要求的功能再此程序中已全部实现，并在此基础上更加完善，而且也拓展更多新的功能。但是在写这些代码的过程中遇到过诸多问题，例如，函数传参错误，变量名写错，空间未开辟和未释放等问题。

虽然问题诸多，但是经过不断调试测试终于完善了这些错误。解决一些错误必须要精确到每一步上去，可以使用加断点的方法，也可以使用内置判断错误的方法输出错误问题。算法改进的方向是在数据这方面，如果数据量庞大的话，可能程序会出现宕机状态，因此，解决数据存储时的速度慢等问题。

在开发校园导航系统的过程中，不仅锻炼了技术能力，还培养了团队协作和问题解决的能力。与其他开发人员和用户进行交流和合作，共同完善导航系统，为校园带来更便利的服务。校园导航系统可以为学生、教职员工及访客提供方便的定位和导航服务，减少迷路和浪费时间的情况。我非常享受这个项目带来的挑战和成就感，同时也深刻意识到设计和技术对于提供优质用户体验的重要性。

9. 参考文献

[1] 严蔚敏.数据结构（C 语言版）[M]. 北京：清华大学出版社, 2018.

[2] 谭浩强.C 语言程序设计[M]. 北京：清华大学出版社, 2018.

[3] 李冬梅.数据结构习题解析与实验指导[M]. 北京：人们邮电出版社,2017.

[4] 张晓明.简明 C++面向对象设计与实践[M]. 郑州：河南大学出版社，2021.

[5] 殷人昆等.数据结构——用面向对象方法与 C++描述[M]. 北京：清华大学出版社，2021.