



燕山大学

《数据结构与算法》三级项目报告

YSU 地图导航

——有权无向图的应用

姓 名	乔翱	李华宪	张伟超	王响
学 号	201811040809	201811040807	201811040779	201811040794
成 绩				
指导老师	窦燕			
时 间	2020 年 1 月 1 日星期三			

2019 年 9 月

摘要

该项目报告分为摘要，前言，目录，正文，结论，参考文献，和项目小组工作量和成绩表自查七部分；

摘要简述了项目报告的构成和内容，结论表达了我们对于这次三级项目的心得体会；

正文中包含研究内容的基本原理，所采用的研究方法及相关工具，项目的方案设计和研究结果并讨论四部分；

正文中在解释原理的同时，还穿插了我们组的核心代码和项目运行截图。

目录

摘要.....	3
目录.....	4
前言.....	5
正文.....	8
1.1 研究内容的基本原理.....	8
1.2 所采用的研究方法及相关工具.....	9
1.3 项目的方案设计.....	10
1.4 研究结果并讨论.....	10
结论.....	24
参考文献.....	25
三级项目小组工作量及成绩表自查.....	26

前言

项目研究报告的目的

为了展示和汇报我们小组对于地图导航功能的了解和研究，以及对于我们燕山大学地图导航这个项目的具体完成情况。

项目研究报告的范围

我们的项目研究报告是基于地图导航来进行撰写的，主要是介绍当前相关领域所做的工作和研究的概况，以及我们组成员对这个地图导航的具体实现方法和具体的实现情况，以及介绍我们组的分工情况和组内互评。

相关领域所做的工作

桌面 web 地图发展较早，基本实现了二维电子地图和影像图，绝大部分地图支持二次开发，在这类地图中，Google 发展较早，也相对比较成熟，智能手机发展起来之后，手机地图也应运而生，手机端的地图也拥有了导航的功能，但是，当下情况下的地图导航在室内定位缺乏了相关技术支持，地图模型也有待完善，所以，其发展也受到了限制。

相关领域研究的概况

(1) 导航地图 1.0,如二维电子地图、遥感影像数据等,隶属于导航地图 1.0 的范畴。这类地图应用中包含了基本的导航定位服务功能,可满足相关生产活动的开、展要求;

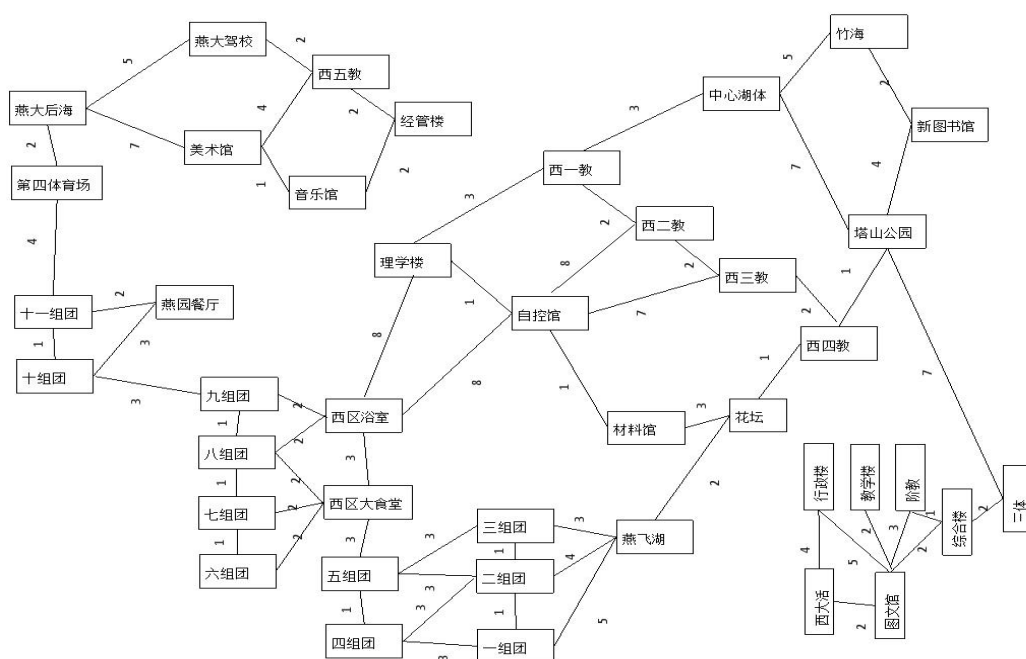
(2)导航地图 2.0。这类地图是在导航地图 1.0 的基础上发展而来的，除具有导航地图 1.0 的所有服务功能以外，还能为使用者提供三维地图、网络云导航、精度良好的导航数据等。现阶段导航地图 2.0 相比导航地图 1.0 在成熟度方面需要进一步提高,与之相关的理论研究力度有待加大。同时,当前某些导航地图产品因具有了导航地图 2.0 的某些功能，如高精度数据的提供、二维矢量地图等，使得这类产品推广应用中的优势增加，且为导航地图未来的更好发展打下了基础。

研究报告的意图

项目研究报告的意图是阐述我们这个项目的分工，概况，具体实现情况及了解当下世界对于这个领域研究的一个深度，广度，和具体方向还有未来关于地图这个方面的发展方向。

预期的结果

关于燕山大学地图导航这个项目，我们的预期结果是可以实现用户输入一个起点，输入一个终点，我们便能给出一个最优最短路径，比如：三组团→西区大食堂→西区浴池→广源超市,最短距离为 2KM;



项目组分工

乔翱：实现基类中的虚函数，以及邻接数组的表示，并检测代码正确性，PPT 的部分制作，项目报告的部分撰写

张伟超：效率检测。需阅读课本第四章内容，严格按照要求正确检测，PPT 的部分制作，项目报告的部分撰写

李华宪：确定一个图是否具有环路，实现 BFS 和 DFS。PPT 的部分制作，项目报告的部分撰写

王响：迪杰斯特拉算法和弗洛伊德算法的代码实现，PPT 的部分制作，项目报告的部分撰写

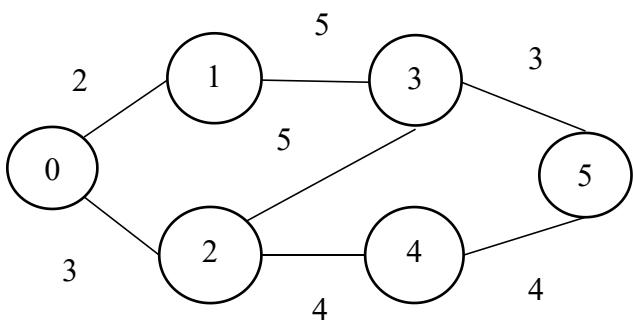
正文

1.1 研究内容的基本原理

无向图：边没有方向的图称为无向图。

加权图：用数学语言讲，设 G 为图，对图的每一条边 e 来说，都对应于一个实数 $W(e)$ （可以通俗的理解为边的“长度”，只是在数学定义中图的权可以为负数），我们把 $W(e)$ 称为 e 的“权”。把这样的图 G 称为“加权图”。

- 1) 继承和实现抽象类 graph。
- 2) 用邻接数组来表示无向有权图。实例：



vector 邻接数组存储的无向有权图形式：

0	1	2	2	3			
1	0	2	3	5			
2	0	3	3	5	4	4	
3	1	5	2	5	5	3	
4	2	4	5	4			

5	3	3	4	4	
---	---	---	---	---	--

一个 node 类型的节点：

weight	next
--------	------

数组中的每一个节点都包含两个值：weight 和 next。

3) 编写函数通过 DFS 的方法判断一个无向图是否存在一个环路。

4) 利用队列的思想实现广度优先遍历。

5) 利用栈的思想深度优先遍历。

6) 迪杰斯特拉算法的实现：迪杰斯特拉最朴素的思想就是按长度递增的次序产生最短路径。即每次对所有可见点的路径长度进行排序后，选择一条最短的路径，这条路径就是对应顶点到源点的最短路径。

7) 弗洛伊德算法的实现：不断地遍历每一个点，并且以每一个点作为中转点看看他的只会不会变化，就可以得到从一个点到任何一个点的最短路径，也就是多源最短路，这就是弗洛伊德算法。

1.2 所采用的研究方法及相关工具

1) 采用的研究方法：

①通过查阅《数据结构、算法与应用》、《大话数据结构》等书籍

②通过浏览网上的博客。

③求助同学、老师。

2) 相关工具：Visual Studio

1.3 项目的方案设计

我们组研究开发的无向有权图燕山大学地图导航目的是帮助同学们解决到达某地的最短路径，避免多走浪费时间精力的问题。

首先我们继承及实现抽象类 graph。然后将地图中的地点和道路抽象成顶点和边的关系存储在 vector 实现的邻接数组中，有了邻接数组便等于有了地图通过深度优先遍历我们可以遍历图中的所有点，通过广度优先遍历可以知道一个地点可以到达的所有地点。我们可以利用迪杰斯特拉算法求出一个地点到其他任何顶点的最短距离，通过把迪杰斯特拉写入循环实现每个地点到其他所有顶点的最短路径，当然我们可以利用更加优雅的弗洛伊德算法去求出每个顶点到其他顶点的最短距离。

- 1) 继承及实现抽象类 graph。
- 2) 邻接数组存储无向有权图。
- 3) 判断是否有环。
- 4) BFS 的实现。
- 5) DFS 的实现。
- 6) 迪杰斯特拉算法的实现。
- 7) 弗洛伊德算法的实现。
- 8) 主界面的设计。

1.4 研究结果并讨论

- 1) 继承及实现抽象类 graph。

将 insertEdge()//插入边、eraseEdge()//删除边、output()//输出邻接数组等函数进行重载。其中我们组对 output() 函数的重载实现了将我们利用 vector 存储图的结构进行输出，方便老师检验。

```
void eraseEdge(int x, int y)
{
    // Delete the edge (i,j).
    if (x < 0 || x >= n || y < 0 || y >= n
```

```

|| !existsEdge(x, y)) {
    return;
}
vector<node>::iterator it;
for (it = a[x].begin(); it != a[x].end(); it++) {
    if (it->next == y) {
        e--;
        a[x].erase(it);
        return;
    }
}
for (it = a[y].begin(); it != a[y].end(); it++) {
    if (it->next == x) {
        e--;
        a[y].erase(it);
        return;
    }
}
}

void insertEdge( edge & theEdge) {
    int v1 = theEdge.x;
    int v2 = theEdge.y;
    int w = theEdge.weight;
    if (v1 < 0 || v2 < 0 || v1 > n || v2 > n || v1 == v2)
    {
        ostringstream s;
        s << "(" << v1 << "," << v2
            << ") is not a permissible edge";
        throw illegalParameterValue(s.str());
    }
    if (v1 < 0 || v1 >= n || v2 < 0 || v2 >= n) {
        return;
    }
    if (!existsEdge(v1,v2)) {
        e++;
        node tem;
        tem.next = v2;
        tem.weight = w;
        a[v1].push_back(tem);
        node tem2;
        tem2.next = v1;
        tem2.weight = w;
        a[v2].push_back(tem2);
    }
    else {
        for (int i = 0; i < a[v1].size(); i++) {
            if (a[v1][i].next == v2) {
                a[v1][i].weight = w;
                break;
            }
        }
    }
}

```

```

        for (int i = 0; i < a[v2].size(); i++) {
            if (a[v2][i].next == v1) {
                a[v2][i].weight = w;
                break;
            }
        }
    }
}

void output(ostream& out) const
{
    // Output
    out << "图的邻接数组为: " << endl;
    for (int i = 0; i < n; i++)
    {
        out << i << " ";
        for (int j = 0; j < a[i].size(); j++)
        {
            out << a[i][j].next << ", " << a[i][j].weight << "
";
        }
        out << endl;
    }
}

```

2) 邻接数组存储无向有权图的点、边及边的长度。

我们组采用的是 vector 二维数组，利用其长度可变性以及 vector.size() 等已有函数简化其存储及遍历过程。数组中的每一个元素都是 node 类型的对象，每个 node 类型的对象都包含两个私有数据项 weight 和 next。第一列元素表示图的顶点按非递减的方式排列，每一行表示该行首元素中存储的顶点所连接的所有顶点及相应边的权值。

3) 判断是否有环。

```

bool circle()
{
    int *prenode = new int[MaxNode];
    int visited[MaxNode], s_top;
    for (int i = 0; i < MaxNode; i++)
    {
        prenode[i] = -1;
        visited[i] = 0;
    }
    stack<int> s;
    s.push(0);
}

```

```

while (!s.empty())
{
    s_top = s.top();
    visited[s_top] = 1;
    s.pop();
    for (int i = 0; i < a[s_top].size(); i++)
    {
        if (!visited[a[s_top][i].next])
        {
            visited[a[s_top][i].next] = 1;
            s.push(a[s_top][i].next);
            prenode[a[s_top][i].next] = s_top;
        }
        else if (a[s_top][i].next != prenode[s_top])
            return true;
    }
}
return false;
}

```

利用 DFS 方法，不同的是要声明一个前驱数组初始化为-1，每访问一个节点就要将前驱数组的值更新为前驱节点所对应的数字，当访问节点所能延展到的顶点除它的前驱节点外还有其他已经被访问到的节点时则终止算法，有环返回 true。如若在遍历完所有的节点时，都没有出现这种情况则该无向图没有环返回 false。

4) BFS 的实现

```

void bfs( int start)
{
    int num = 0;
    bool ver[200] = { 0 };
    queue<int>q;
    ver[start] = 1;
    q.push(start);
    while (!q.empty())
    {
        int u = q.front();
        q.pop();
        if (num == n-1)
            cout << u;
        else
            cout << u<< "->";
        num++;
        for (int i = 0; i < a[u].size(); i++)
        {
            if (ver[a[u][i].next] == 0)
            {
                ver[a[u][i].next] = 1;
            }
        }
    }
}

```

```

        q.push(a[u][i].next);
    }
}
}
}

```

我们利用队列的方法实现 BFS，广度优先遍历的关键在于标记数组以及队列。初始化标记数组为 0 表示所有的顶点都没被访问过，每当访问一个节点时，将其对应的标记数组中的元素设为 1 表示已被访问。通过标记数组判断延展节点是否入队列，当其为 0 时，将顶点入队列并更新其标记值为 1。若为 1 则不入队列。通过判断队列是否为空进而判断是否终止循环。若为空终止循环，算法结束。

5) DFS 的实现。

```

void dfs( int start)
{
    int num = 1;
    int visited[MaxNode], s_top;
    for (int i = 0; i < MaxNode; i++)
    {
        visited[i] = 0;
    }
    visited[start] = 1;
    stack<int>s;
    cout << start << "->";
    for (int i = 0; i < a[start].size(); i++)
    {
        visited[a[start][i].next] = 1;
        s.push(a[start][i].next);
    }
    while (!s.empty())
    {
        s_top = s.top();
        visited[s_top] = 1;
        if (num == n - 1)
            cout << s_top;
        else
            cout << s_top << "->";
        num++;
        s.pop();
        for (int i = 0; i < a[s_top].size(); i++)
        {
            if (!visited[a[s_top][i].next])
            {

```

```

        visited[a[s_top][i].next] = 1;
        s.push(a[s_top][i].next);
    }
}
}
}

```

深度优先遍历要用到栈的思想，其关键在于标记函数和栈。初始化标记数组为 0 表示所有的顶点都没被访问过，每当访问一个节点时，将其对应的标记数组中的元素设为 1 表示已被访问。通过标记数组判断延展节点是否入栈，当其为 0 时，将顶点入队列并更新其标记值为 1。若为 1 则不入栈。通过判断栈是否为空进而判断是否终止循环。若为空终止循环，但是还要判断标记数组是否都为 1，如果都为 1 则算法结束，反之则选一个标记数组为 0 的顶点入栈开始循环，直至所有的顶点都被访问过即标记数组均为 1 时算法结束。

6) 迪杰斯特拉算法的实现

```

void Dijkstra(int start, int end) {

    /*int map[maxn][maxn];*/
    int dis[maxn];

    int vis[maxn]; //记录更新过的点
    //初始化
    memset(path, -1, sizeof(path));
    memset(dis, 0x3f, sizeof(dis)); //初始化为无穷大
    memset(vis, 0, sizeof(vis));
    dis[start] = 0; //自身到自身的距离为 0
    int num = 0;
    int k = 0;
    while (1)
    {
        k = 0;
        for (int j = 0; j < n; j++)
        {
            if (!vis[j] && !vis[k]) //找未收录顶点中 dis 值最小的
                k = dis[j] < dis[k] ? j : k; //这里第一次找到的是
起点

            else if (vis[k])
            {
                k++;
            }
        }
    }
}

```

```

    }
}
//没有未收录的点，则返回
vis[k] = 1;
num++;
for (int j = 0; j < a[k].size(); j++)
{
    //第一次循环只有起点的邻接点距离被更新，每次都更新新找
    到的点的邻接点
    if (dis[a[k][j].next] > dis[k] +
a[k][j].weight&&!vis[a[k][j].next])
    {
        dis[a[k][j].next] = dis[k] +
a[k][j].weight;
        path[a[k][j].next] = k; //路径被改变，重新记
        录前驱，最短路是由最短路+某一条固定路组成，所以前驱是有效的
    }
}
if (num==n) break;
}
print(path[end]);
printf("%d\n", end);
//打印最短距离
printf("%d\n", dis[end]);
}

```

迪杰斯塔拉最朴素的思想就是按长度递增的次序产生最短路径。即每次对所有可见点的路径长度进行排序后，选择一条最短的路径，这条路径就是对应顶点到源点的最短路径。单源最短路径需要三个数组，分别为标记数组、最短路径权值数组以及前驱数组，还有一个已知数据即起点。标记数组判断顶点是否被访问过，最短路径权值数组中存储的是起点到其它各点的最短路径权值，前驱数组中存储的是起点到各点最短路径上的中间结点。不断遍历最短路径权值数组找到权值最小并且没被访问过的节点，查找其延展节点（即该顶点能都到达同时没被访问过的顶点）比较路径权值的大小，更新最短路径权值及前驱节点。直到所有的顶点都被访问。

7) 弗洛伊德算法的实现

```

void floyd(vector<vector<int>> &distmap, //可被更新的邻接矩阵，更新后
不能确定原有边
    vector<vector<int>> &path) {
    const int &NODE = distmap.size(); //用邻接矩阵的大小传递顶点
    个数，减少参数传递

```



```

        path.assign(NODE, vector<int>(NODE, -1)); //初始化路径数组
        for (int k = 1; k != NODE; ++k) //对于每一个中转点
            for (int i = 0; i != NODE; ++i) //枚举源点
                for (int j = 0; j != NODE; ++j) //枚举终点
                    if (distmap[i][j] > distmap[i][k] +
distmap[k][j]) //不满足三角不等式
                        {
                            distmap[i][j] = distmap[i][k] +
distmap[k][j]; //更新
                            path[i][j] = k; //记录路径
                        }
    }

```

需要初始化两个矩阵，分别为前驱矩阵 P 和权值矩阵 D，需要三层循环第一层对中间结点 k 的循环，第二层则是对起点 i 的遍历，第三层是对终止节点 j 的遍历。

循环体：①如果经过下标为 k 顶点路径比原两点间路径更短。

②将当前两点间权值设为更小的一个。

③路径设置经过下标为 k 的顶点。

8) 主界面的设计。

主界面是用户和程序设计人员的交流窗口，为方便用户理解使用，我们将地图中所包含的地点及其在邻接数组中对应的数字符号一一对应的打印在主界面，最重要的就是我们的功能要陈列在主界面以达到方便用户使用的目的，同时我们在用户每做出一步操作时都给出相应的提示，指导用户正确的使用该程序。

部分指导代码如下：

```

#include "arrayWGraph.h"
#include "test.h"

int main() {
    creategraph();
    int something = 0;

    while (something != -1)
    {

```

```

        system("cls");
        if (something == 0)
        {
            system("cls");
            cout << "*****菜单
*****" << endl;
            for (int i = 0; i <= 40; i++)
            {
                cout << i << ":" << s[i]<<" ";
                if ((i+1) % 5 == 0)
                    cout << endl;
            }
            cout << endl;
            cout << "1、测试该图是否是有权无向图" << endl;
            cout << "2、测试是否有这条边" << endl;
            cout << "3、广度优先搜索该图" << endl;
            cout << "4、深度优先搜索该图" << endl;
            cout << "5、节点个数" << endl;
            cout << "6、边个数" << endl;
            cout << "7、删除一条边" << endl;
            cout << "8、插入一条边" << endl;
            cout << "9、从 A 到 B 的最短路径" << endl;
            cout << "10、测试该图是否含有环" << endl;
            cout << "11、输出邻接数组" << endl;
            cout << "12、所有地方之间的最短距离" << endl;
            cout << "请输入您所想要实现的功能前的序号" <<
endl;

            cin >> something;
        }
        if (something == 1)
        {
            system("cls");
            bool flag = mygraph.weighted();
            if (flag)
                cout << "This is a weighted graph" <<
endl;

            else
                cout << "This is not a weighted graph"
<< endl;

            flag = mygraph.directed();
            if (flag)
                cout << "This is a directed graph" <<
endl;

            else
                cout << "This is a undirected graph" <<
endl;

        }
        if (something == 2)
        {
            system("cls");

```

```

        int i, j;
        cout << "cin the vertex of the edge" << endl;
        cin >> i >> j;
        bool flag = mygraph.existsEdge(i, j);
        if (flag)
            cout << "exsit the edge of " << i << "-"
<< j << endl;
        else
            cout << "Not exsit the edge of " << i <<
            "-" << j << endl;
    }
    if (something == 3)
    {
        system("cls");
        cout << "cin the start place" << endl;
        int start;
        cin >> start;
        cout << "图的广度优先搜索—";
        mygraph.bfs(start);
        cout << endl;
    }
    if (something == 4)
    {
        system("cls");
        cout << "cin the start place" << endl;
        int start;
        cin >> start;
        cout << "图的深度优先搜索—";
        mygraph.dfs(start);
        cout << endl;
    }
    if (something == 5)
    {
        system("cls");
        int numofv = mygraph.numberOfVertices();
        cout << "点的数量为—" << numofv << endl;
    }
    if (something == 6)
    {
        system("cls");
        int numofe = mygraph.numberOfEdges();
        cout << "边的数量为—" << numofe << endl;
    }
    if (something == 7)
    {
        system("cls");
        int first, second;
        cout << "please cin the vertex of the edge"
<< endl;
        cin >> first >> second;

```

```

        mygraph.eraseEdge(first, second);
    }
    if (something == 8)
    {
        system("cls");
        int first, last, theweight;
        cout << "please cin the first last weight" <<
endl;

        edge e;
        cin >> e.x>> e.y >> e.weight;
        try
        {
            mygraph.insertEdge(e);
        }
        catch (illegalParameterValue)
        {
            cout << "It is not a permissible edge"
<< endl;

            system("pause");
            return 0;
        }
    }
    if (something == 9)
    {
        system("cls");
        int start, destination;
        cout << "cin the start and the destination"
<< endl;

        cin >> start >> destination;
        mygraph.Dijkstra(start, destination);
    }
    if (something == 10)
    {
        system("cls");
        bool flagloop = false;
        flagloop = mygraph.circle();
        if (flagloop == true)
        {
            cout << "This graph has a loop" << endl;
        }
        else
            cout << "This graph doesn't have a loop"
<< endl;
    }
    if (something == 12) {
        system("cls");
        mygraph.shortallpath();
    }
    if (something == 11)
    {
        system("cls");

```

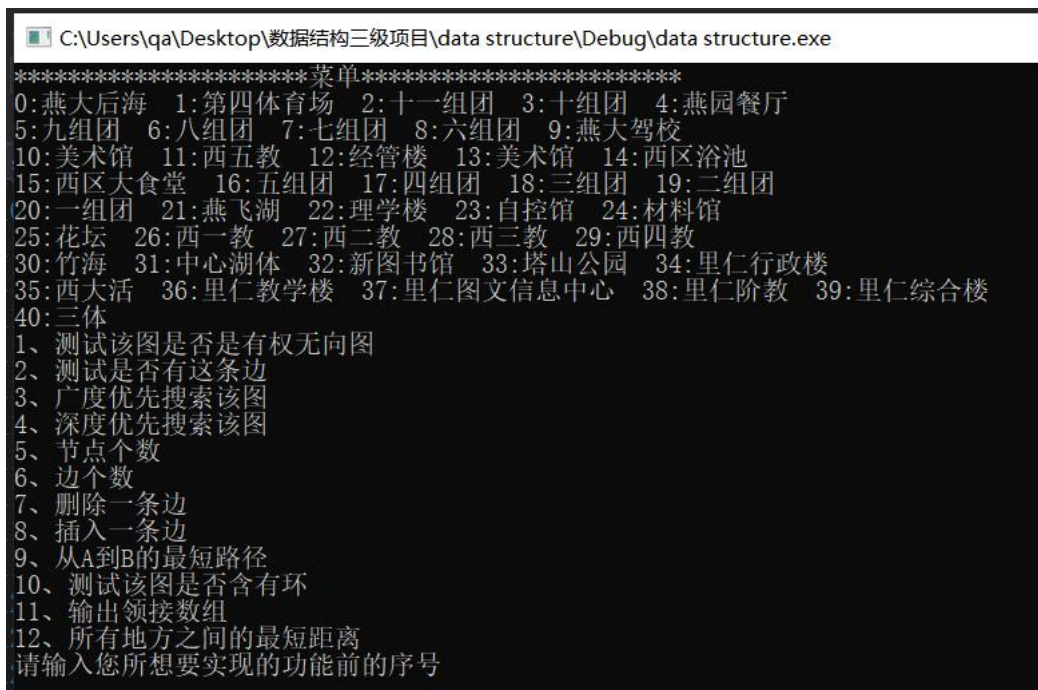
```

        mygraph.output(cout);
    }
    cout << "press 0 :return menu" << endl;
    cout << "press -1:exit" << endl;
    cin >> something;
}
cout << "*****  **      **          **          *****
**      **      **" << endl;
cout << "      **      **      **          **      **          **
**      **      ** " << endl;
cout << "      **      **      **          **          **          **
**      **      **" << endl;
cout << "      **          *****          *****          **      **
**      ***" << endl;
cout << "      **      **      **      **          **      **      **
**      **      **" << endl;
cout << "      **      **      **      **          **      **
*****      **      **" << endl;

    return 0;
}

```

主界面效果图:



```

C:\Users\qa\Desktop\数据结构三级项目\data structure\Debug\data structure.exe
*****菜单*****
0:燕大后海 1:第四体育场 2:十一组团 3:十组团 4:燕园餐厅
5:九组团 6:八组团 7:七组团 8:六组团 9:燕大驾校
10:美术馆 11:西五教 12:经管楼 13:美术馆 14:西区浴池
15:西区大食堂 16:五组团 17:四组团 18:三组团 19:二组团
20:一组团 21:燕飞湖 22:理学楼 23:自控馆 24:材料馆
25:花坛 26:西一教 27:西二教 28:西三教 29:西四教
30:竹海 31:中心湖体 32:新图书馆 33:塔山公园 34:里仁行政楼
35:西大活 36:里仁教学楼 37:里仁图文信息中心 38:里仁阶教 39:里仁综合楼
40:三体
1、测试该图是否是有权无向图
2、测试是否有这条边
3、广度优先搜索该图
4、深度优先搜索该图
5、节点个数
6、边个数
7、删除一条边
8、插入一条边
9、从A到B的最短路径
10、测试该图是否含有环
11、输出邻接数组
12、所有地方之间的最短距离
请输入您所要实现的功能前的序号

```

图以邻接数组形式存储的结构输出：

```
C:\Users\qa\Desktop\数据结构三级项目\data
图的邻接数组为：
0 1, 2 9, 5 10, 7
1 0, 2 10, 3 13, 8 2, 4
2 3, 1 4, 2 1, 4
3 2, 1 4, 3 5, 3
4 2, 2 3, 3
5 3, 3 6, 1 14, 2
6 5, 1 7, 1 14, 2 15, 2
7 6, 1 8, 1 15, 2
8 7, 1 15, 2
9 0, 5 11, 2
10 0, 7 1, 3 13, 1 11, 4
11 10, 4 9, 2 12, 2
12 11, 2 13, 2
13 1, 8 10, 1 12, 2
14 6, 2 5, 2 15, 3 22, 8 23, 8
15 14, 3 8, 2 7, 2 6, 2 16, 3 24, 10
```

判断是否有环：

```
C:\Users\qa\Desktop\数据结构三级项目
This graph has a loop
press 0 :return menu
press -1:exit
```

广度优先遍历：

```
C:\Users\qa\Desktop\数据结构三级项目\data structure\Debug\data structure.exe
cin the start place
4
图的广度优先搜索——4->2->3->1->5->0->10->13->6->14->9->11->12->7->15->22->23->8->16->24->26->27->28->17->18->19->25->29->31->20->21->33->30->35->32->36->40->34->37->38->39
press 0 :return menu
press -1:exit
```


深度优先遍历：

C:\Users\qa\Desktop\数据结构三级项目\data structure\Debug\data structure.exe

cin the start place

6

图的深度优先搜索——6->15->24->29->33->40->39->37->35->21->18->19->17->20->34->38->36->31->26->27->22->30->32->28->25->23->16->8->14->7->5->3->4->2->1->13->12->11->9->10->0

press 0 :return menu

press -1:exit

八组团到里仁教学楼的最短路径（迪杰斯特拉算法）：

C:\Users\qa\Desktop\数据结构三级项目\data structure\Debug\data structure.exe

cin the start and the destination

6 34

八组团->西区大食堂->五组团->三组团->燕飞湖->西大活->里仁行政楼

17

press 0 :return menu

press -1:exit

弗洛伊德算法的实现的部分截图：

C:\Users\qa\Desktop\数据结构三级项目\data structure\Debug\data structure.exe

燕大后海->第四体育场最短距离为2, 打印路径: 燕大后海->第四体育场

燕大后海->十一组团最短距离为6, 打印路径: 燕大后海->第四体育场->十一组团

燕大后海->十组团最短距离为7, 打印路径: 燕大后海->第四体育场->十一组团

燕大后海->燕园餐厅最短距离为8, 打印路径: 燕大后海->第四体育场->十一组团

燕大后海->九组团最短距离为10, 打印路径: 燕大后海->第四体育场->十一组团

燕大后海->八组团最短距离为11, 打印路径: 燕大后海->第四体育场->十一组团

燕大后海->七组团最短距离为12, 打印路径: 燕大后海->第四体育场->十一组团

燕大后海->六组团最短距离为13, 打印路径: 燕大后海->第四体育场->十一组团

燕大后海->燕大驾校最短距离为5, 打印路径: 燕大后海->燕大驾校

结论

该项目的重点工作首先是对于有权无向图应用的一个思考，我们刚开始想了两种方案，一个是最小生成树还有一个就是最短路径，我们之所以选择最短路径是因为，他跟我们的生活息息相关而且比较通俗易懂，我们选择做燕山大学地图导航是因为燕山大学真的太绕了，所以我们想结合这个问题做出我们的项目。

其次是对具体实现方法的思考，没错，就是迪杰斯特拉算法和弗洛伊德算法。然后组长对组内各个人的任务进行了具体分工，每个人写一部分代码，然后组长负责最后整合，所以，我们，每个人都很有参与感，因为每个人都写了一部分代码，所以最后项目能够运行并且成功运行的时候，每个人都很开心！

这次的三级项目不再是个人孤军奋战，而是团体合作，共同前进，所以给了我不一样的感觉，我们的团队协作能力得到了提升，也培养了团队之间的默契，整个过程中，组长分配任务，副组长们完成任务，并且能够很坦白的提出自己的意见，我们再进行项目的重新安排，整个过程其乐融融，在一个比较轻松的环境下完成了这次的项目。

参考文献

- [1]沈建新.导航地图发展现状和趋势分析[J].智能城市,2018,4(16):70-71。
- [2]蒋秉川, 杨振发.机器人超高分辨率立体网格导航地图建模研究[J].系统仿真学报, 2017 。
- [3]程杰, 大话数据结构[M], 2011,6。
- [4]Sartaj Sahni,数据结构、算法与应用[M], 2019,5。

其中：M 表示书；J 表示期刊；C 表示会议；D 表示博士硕士论文；

三级项目小组工作量及成绩表自查

姓名	所做的工作	成绩	签字确认
乔翱	实现基类中的虚函数，以及邻接矩阵的表示，并检测代码正确性，PPT 的部分制作，项目报告的部分撰写以及代码的汇总和更改	10	
张伟超	效率检测。需阅读课本第四章内容，严格按照要求正确检测，PPT 的部分制作，项目报告的部分撰写	9	
李华宪	确定一个图是否具有环路，实现 BFS 和 DFS。PPT 的部分制作，项目报告的部分撰写	8	
王响	迪杰斯特拉算法和弗洛伊德算法的代码实现，PPT 的部分制作，项目报告的部分撰写	8	