# 四川师范大学 实验报告

学期： 2024 至 2025 第一学期 实验成绩：

课程名字：《程序设计基础——数据结构(C语言版)》 专业： 信息与计算科学

班级： 2023 级 9 班 实验编号： 08

实验项目： 实验八 指导老师： 冯山

姓名： 刘智恒 学号： 2023060522

**一、实验题目**

图的建立与遍历算法

**二、实验目的及要求**

1.熟练掌握图的邻接矩阵和邻接表的存储方式。

2.熟练掌握图的基本运算算法(如图的遍历)。

3.掌握常见的应用算法，如MST、Top排序、关键路径、最短路径等。

**三、实验内容：(类C算法的程序实现，任选其二)**

1.设计并实现图的基本存储结构和运算算法，包含DFS()和BFS()。(必做)

2.设计并实现MST生成算法(Prim或Kruskal)。

3.设计并实现Top排序算法。

4.设计并实现关键路径算法。

5.设计并实现最短路径算法(Dijkstra或Floyd)。

**四、实验准备**

1.计算机设备;

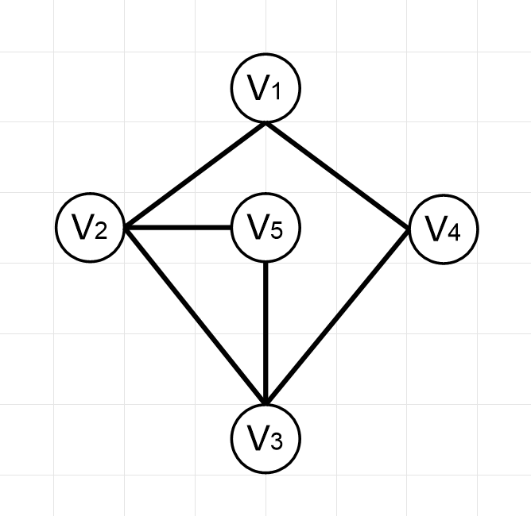
2.程序调试环境的准备，本实验采用**Microsoft Visual Studio**环境;

3.实验内容的算法分析与代码设计与分析准备；

4.实验源程序**Exp\_8(1)**准备。

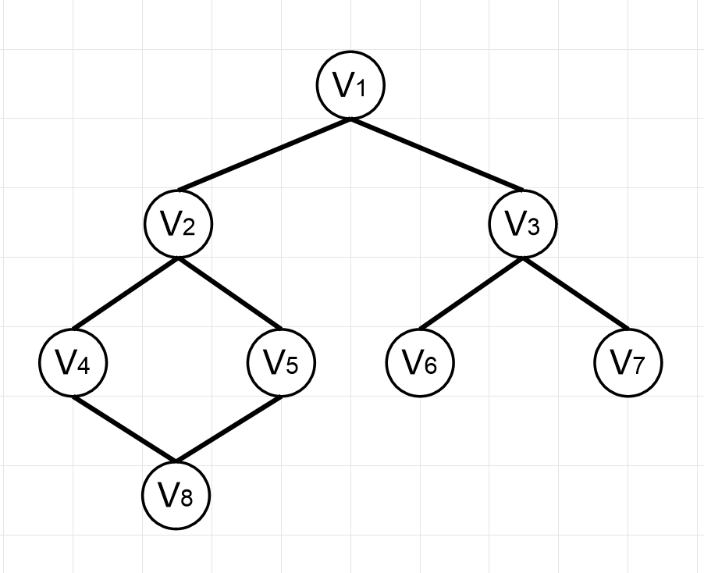
5.实验测试用例：

**（1）无向图1如下图所示：**

****

**图4.1 G1**

**（2）无向图2如下图所示：**

****

**图4.2 G2**

**五、实验内容**

**（一）问题分析**

（1）图的结构比较复杂，任意两个顶点之间都可能存在联系，即图没有顺序映像的存储结构。但可以借助**数组**的数据类型来表示元素之间的关系，以及用**多重链表**表示图。本实验中采用**邻接表**的链式存储结构对**无向图**进行表示。

（2）在邻接表中，对图中**每个顶点建立一个单链表**，第**i**个单链表中的结点表示**依附于当前顶点vi的边**（对有向图是以顶点vi为尾的弧）。其中，每个结点由3个域组成：

**[1]邻接点域（adjVex）**：指示**与顶点vi邻接的点**在图中的位置；

**[2]链域（nextArc）**：指示下一条边或弧的结点；

**[3]数据域（info）**：用于存储和边或弧相关的信息，如**权值（weight）**。

除此之外，每个链表上还**附设一个表头结点**。在表头结点中，除了设有**链域（firstArc）用于指向链表中第一个结点**之外，还设有用于存储顶点vi的名称或其它有关信息的**数据域（data）**。

（3）一个图的邻接表存储结构如下：

**#define MAX\_VERTEX\_NUM 20**

**//边（弧）结点结构**

**typedef struct ArcNode{**

**int adjVex; //该弧所指向的顶点的位置（下标）**

**struct ArcNode\* nextArc; //指向下一条弧的指针**

**//InfoType info; //该弧的相关信息**

**}ArcNode;**

**//顶点结构**

**typedef struct VNode{**

**VertexType data; //顶点信息**

**ArcNode\* firstArc; //指向第一条依附该顶点的弧的指针**

**}VNode, AdjList[MAX\_VERTEX\_NUM]; //（顶点）结构体数组**

**//（无向）图结构**

**typedef struct{**

**AdjList vertices; //顶点数组**

**int vexNum, arcNum; //图的当前顶点数和弧数**

**//int kind; //图的种类标志**

**}ALGraph; //Adjacency List Graph**

（4）各项操作基本分析：

**[1]创建邻接表：**为了存储图，首先需要建立邻接表。邻接表的建立步骤如下：

**<1>**输入总顶点数（**vexNum**）和总弧数（**arcNum**）。

**<2>**创建顶点表：依次输入顶点的信息（**data**）存入顶点表中；使每个表头结点的指针域（**firstArc**）初始化为NULL。

**<3>**创建邻接表：依次输入**每条边依附的两个顶点**（**vi，vj**）；确定两个顶点的序号**i**和**j**（**LocateVex**），**建立边结点**；将此边结点**分别插入到vi和vj对应的两个链表的头部**（**头插法**）

**[2]DFS遍历（Depth-First Search）**：类似于树的**先根遍历**。其基本思想为：

设初始状态是图中所有顶点未曾被访问，然后依次从v的未被访问的邻接点出发深度优先遍历图，**直至图中所有和v有路径（相通）的顶点都被访问**。若此时图中还有顶点未被访问，则**另选图中一个未曾被访问的顶点作为起始点**（**回溯**），重复上述过程，直至图中所有顶点都被访问为止。

显然，这是一个**递归**的过程。为了在遍历过程中便于**区分当前顶点是否已经被访问**，需附设一个**访问标志数组visited[0,…,n-1]**，其初值设置为“**false**”。一旦当前顶点被访问，则其值置为“**true**”。

**[3]BFS遍历（Breadth-First Search）**：类似于树的**按层次遍历**的过程。其基本思想为：

从图中某个顶点**v**出发，在访问了**v**之后，依次访问**v**的**各个未曾访问的邻接点**，然后再分别从这些邻接点出发**依次访问它们的邻接点**，并且使“**先被访问的顶点的邻接点**”**先于**“**后被访问的顶点的邻接点**”被访问，直至图中所有已被访问的顶点的邻接点都被访问到。若此时图中还有顶点未被访问，则另选图中一个未曾被访问的顶点作为起始点，重复上述过程，直至图中所有顶点都被访问到为止。

和深度优先遍历类似，在遍历过程中也需要一个**访问标志数组visited[0,…,n-1]**。并且，为了**顺次访问**路径长度为2,3,…的顶点，需要**附设队列**以存储已被访问的路径长度为1,2…的顶点。

**（二）算法描述**

（1）**Status CreateUDG(ALGraph \*G);**

//创建无向图：首先输入顶点和边的数量，然后输入每个顶点的数据，初始化每个顶点的链表为空；接着输入每条边连接的两个顶点，通过头插法将这两个顶点的边添加到各自的邻接表中。

（2）**int LocateVex(ALGraph G, VertexType v);**

//查找给定顶点值在图中的索引位置。如果找到，返回其索引；否则返回错误标志。

（3）**void Print\_AdjList(ALGraph G);**

//打印邻接表，显示每个顶点及其直接相邻的顶点。

（4）**void DFS(ALGraph G, VertexType v, Status(\*Visit)(VertexType v))**;

//利用递归的方式遍历图，从某个顶点开始访问其未被访问过的相邻顶点，直到无法继续为止；其中使用一个布尔数组 visited 用于记录每个顶点是否被访问过。

（5）**void DFSTraverse(ALGraph G, Status(\*Visit)(VertexType v))**;

//遍历图中所有顶点，确保每个未被访问的顶点都会被 DFS 遍历。

（6）**SqQueue**：实现一个顺序队列，用于广度优先搜索（BFS）。

//包含基本的队列操作：初始化、销毁、判断空、入队和出队。

（7）**void BFSTraverse(ALGraph G, Status(\*Visit)(VertexType v))**;

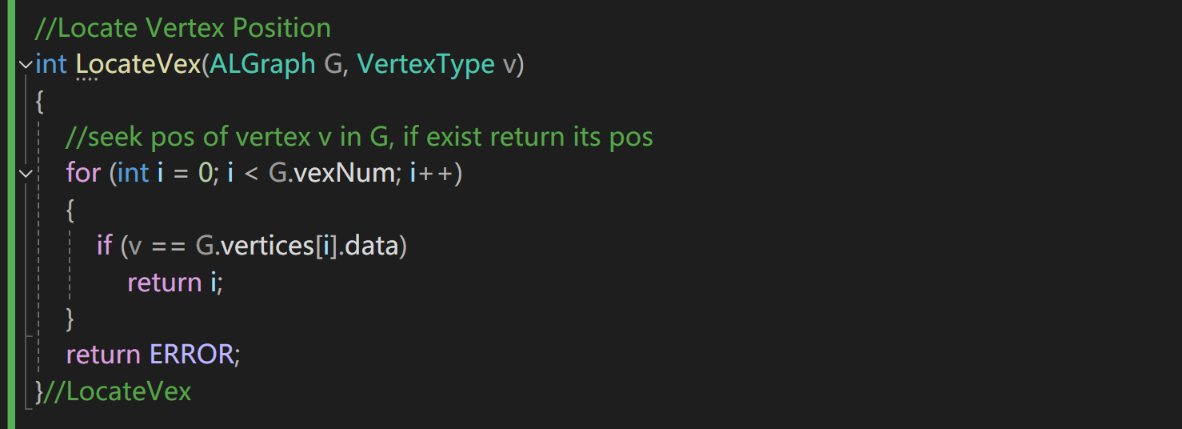
//广度优先搜索：从某个未被访问的顶点开始，访问该顶点，然后将它的所有相邻未被访问的顶点入队。循环执行出队并访问新队列头部的顶点，直到队列为空。

**（三）程序代码**

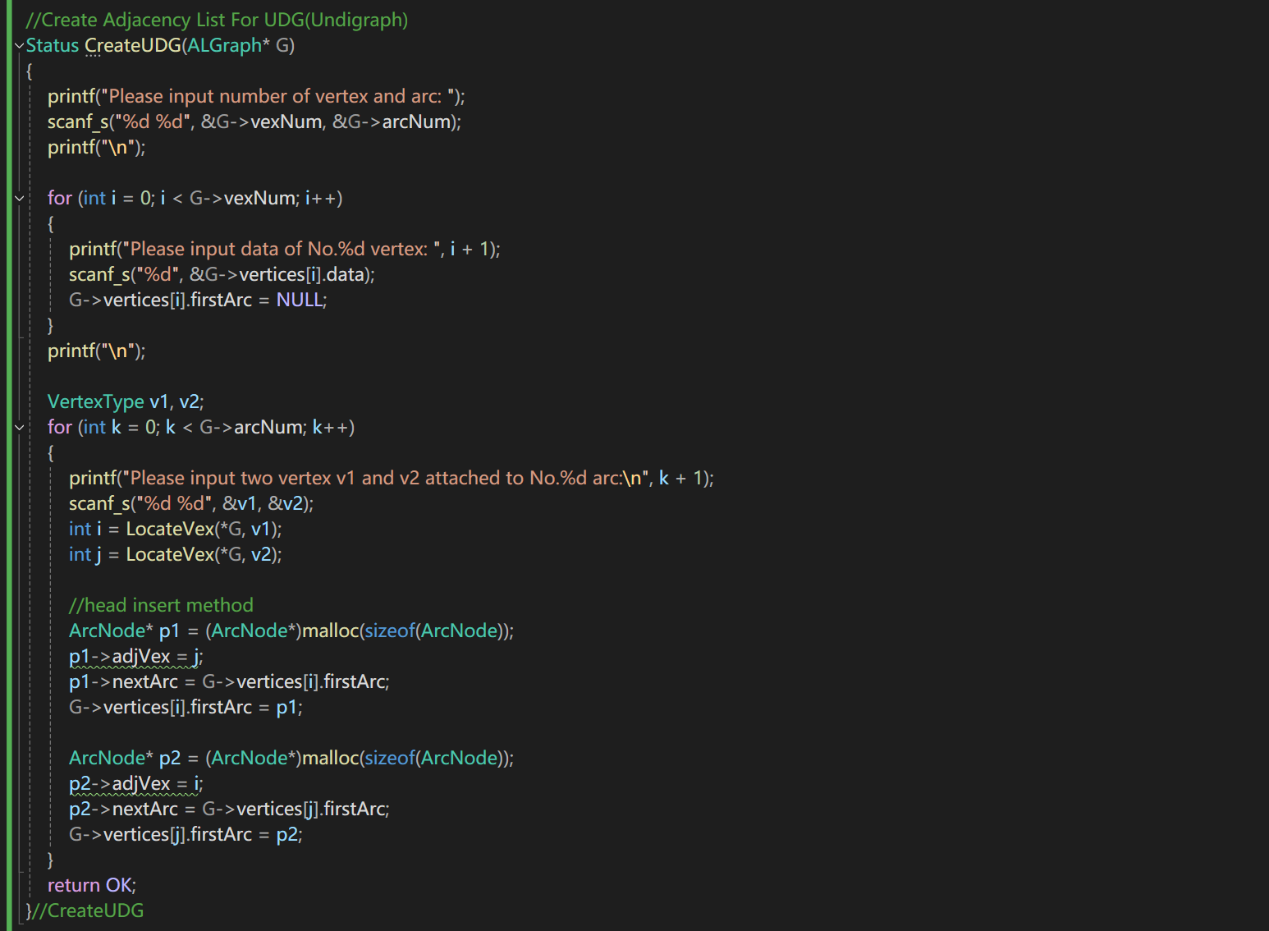
根据以上对**实验内容**的算法描述，求解的程序代码如下：



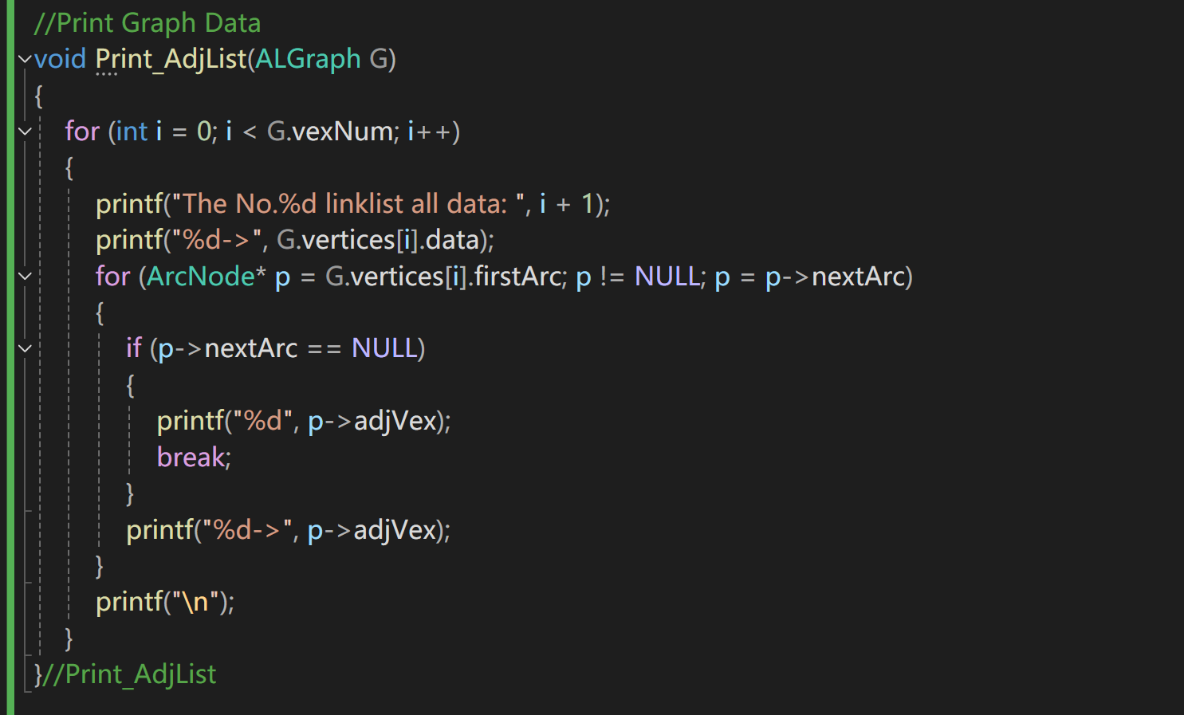
**注：此处只展示主要算法的程序代码图。**

****

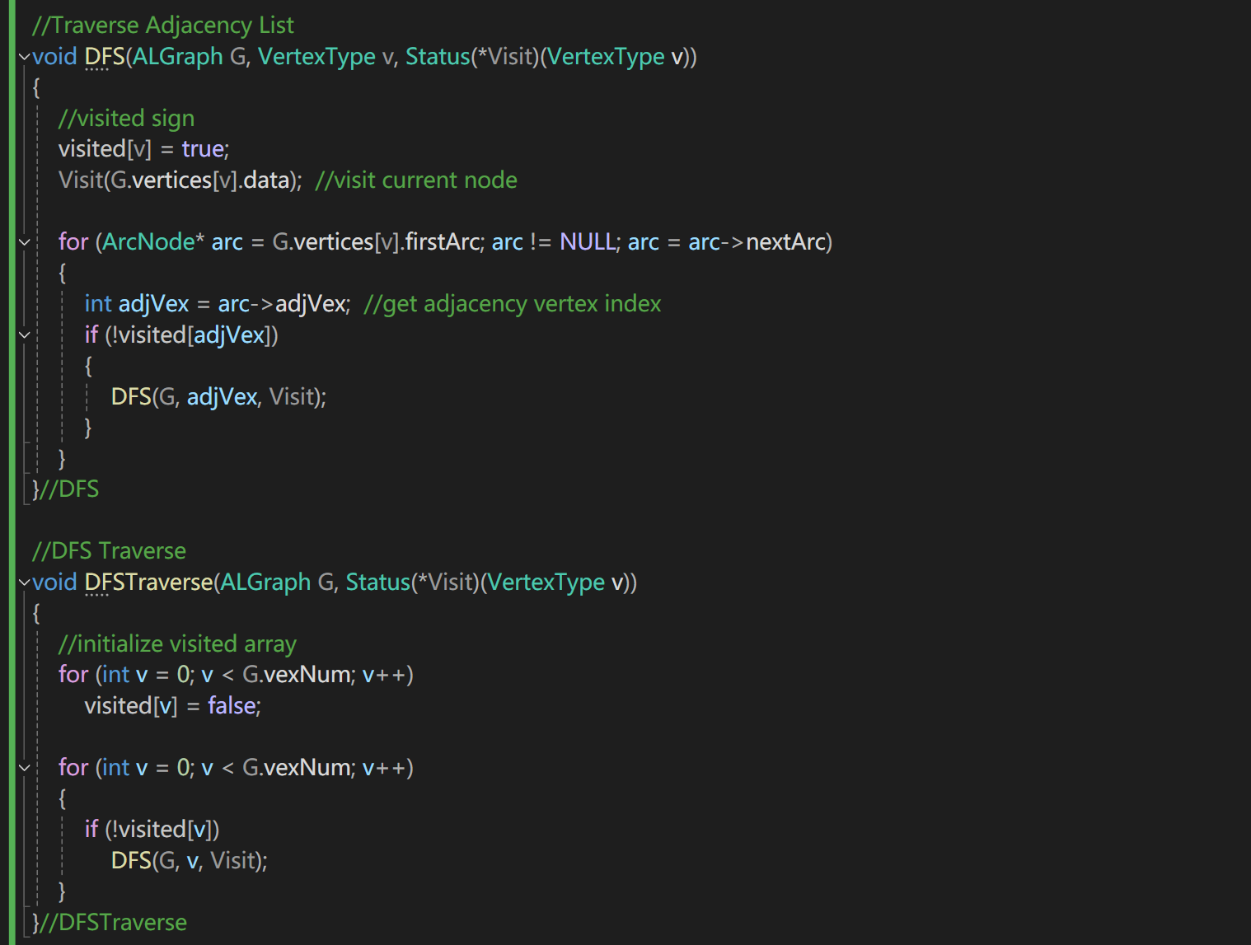
**图5.1确定顶点位置**

****

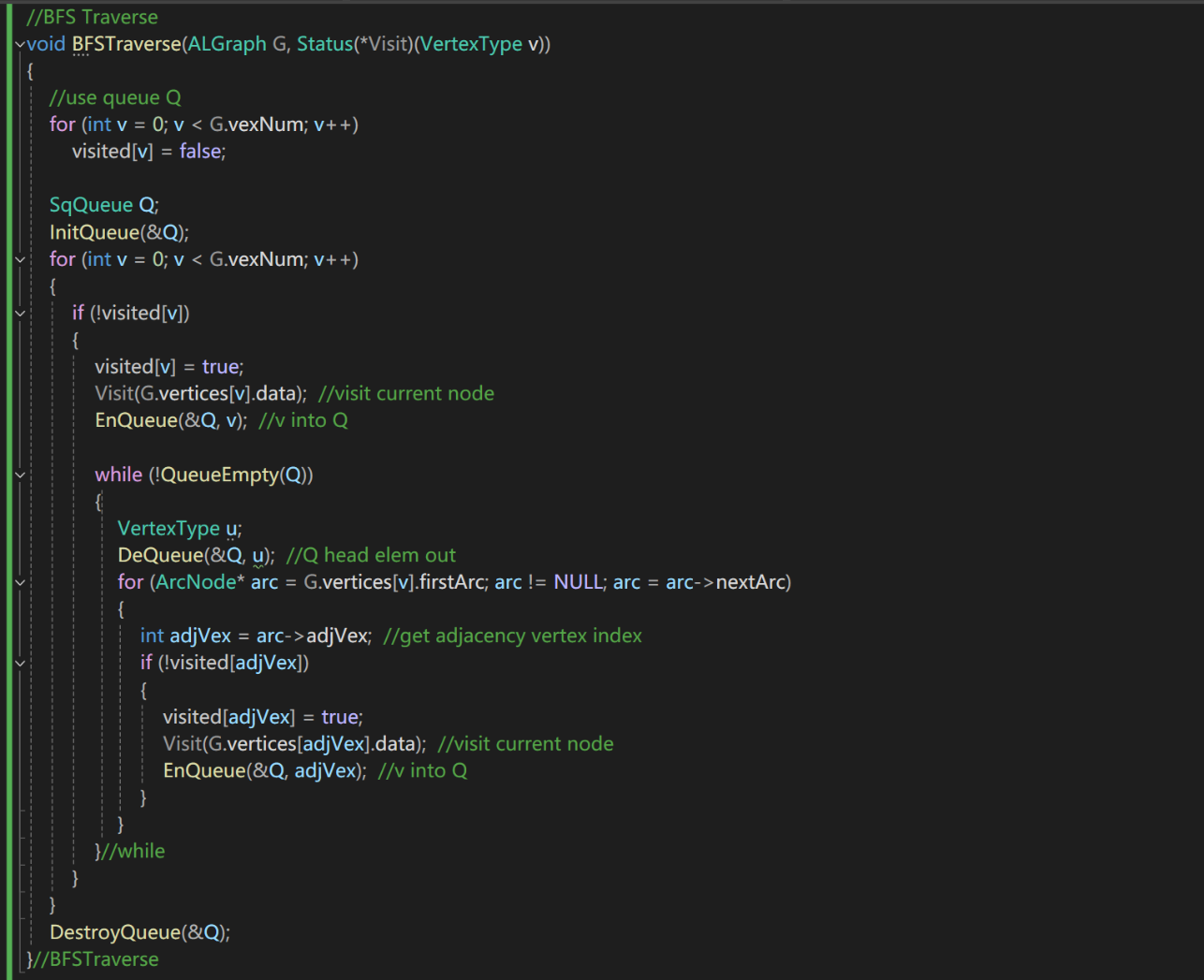
**图5.2创建（无向图）邻接表**

****

**图5.3打印图的数据**

****

**图5.4 DFS遍历图**

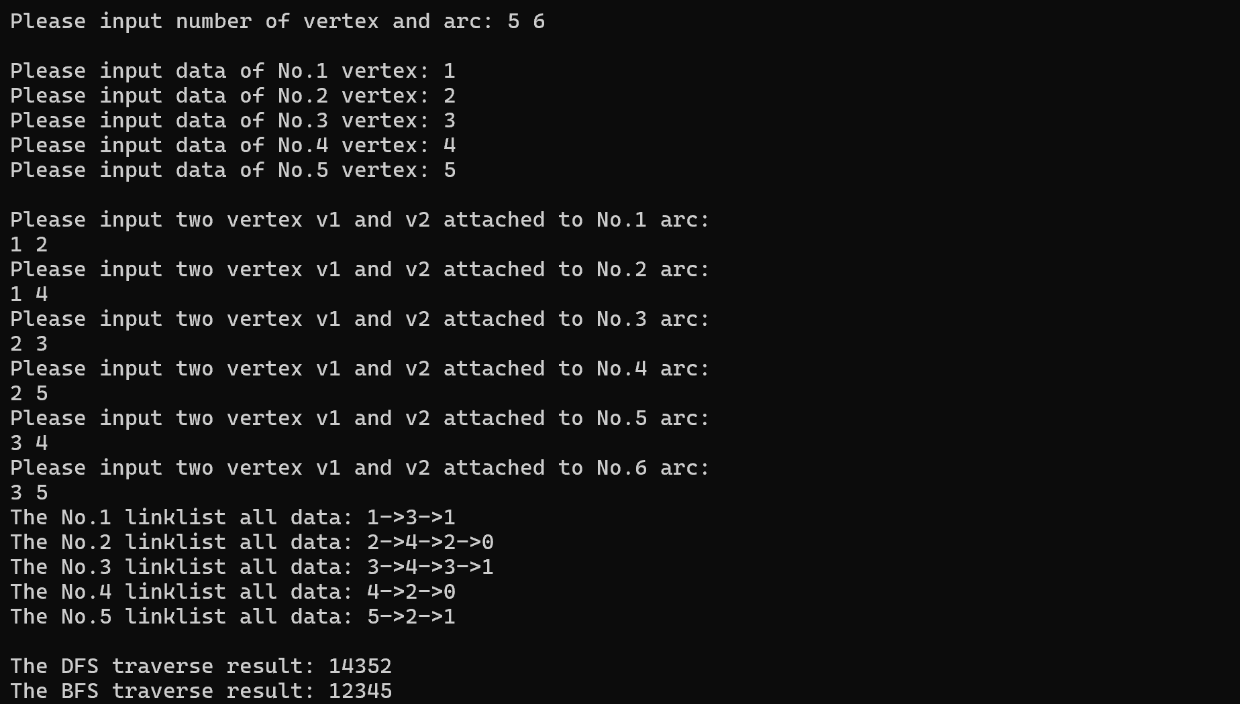
****

**图5.5 BFS遍历图**

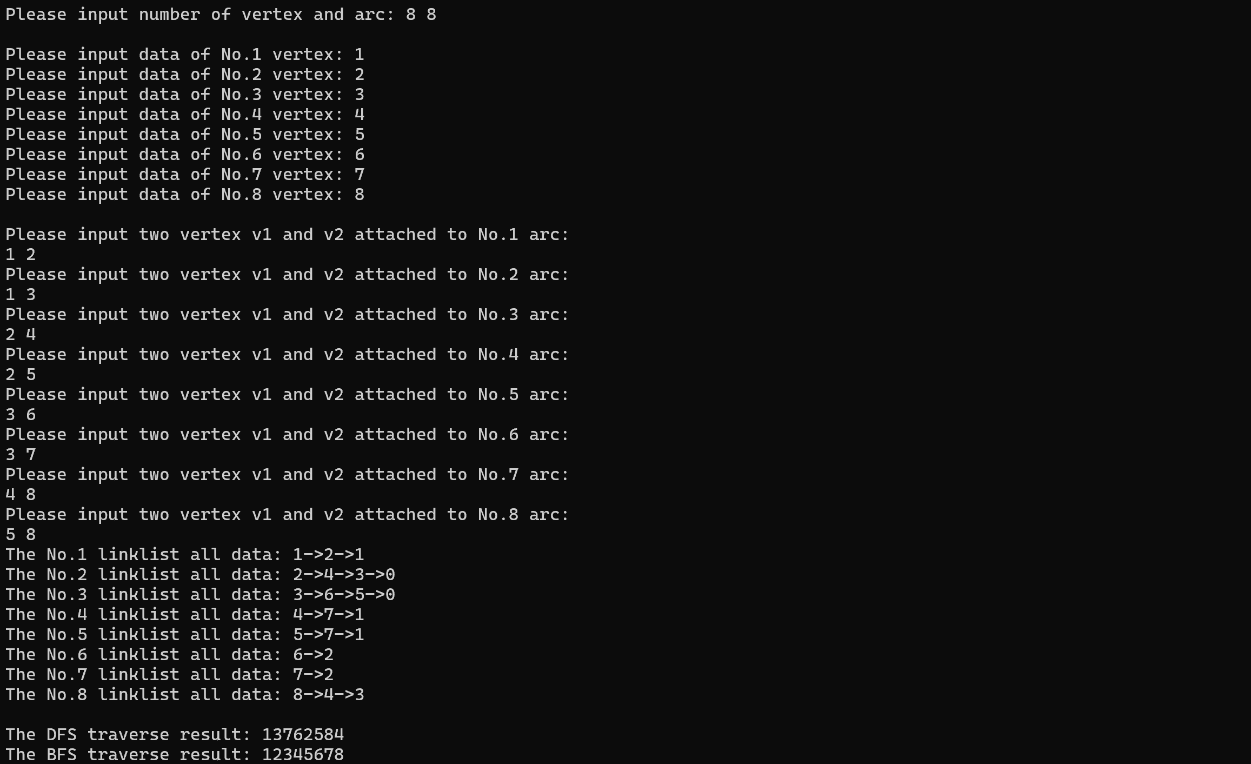
**六、实验结果**

**（一）结果呈现：**

1.根据以上**实验内容**的程序代码运行后进行多次测试，以下是根据**实验内容测试用例**进行测试后的结果示意图：

****

**图6.1实验测试用例（1）结果**

****

**图6.2实验测试用例（2）结果**

**（二）结果分析：**

**1.时间复杂度分析**

**【CreateUDG】—— 创建（无向图）邻接表**

输入顶点数和边数，若设顶点数为**V**，边数为**E**。在创建每个顶点时，都会进行**O(1)**的操作，总共进行**V**次，所以时间复杂度为**O(V)**；在添加每条边时，需要查找两个顶点的索引，然后进行插入操作，插入操作为**O(1)**,因此，添加**E**条边的时间为**O(E)**。故整体时间复杂度为：**O(V+E)**。

**【Print\_AdjList】—— 打印邻接表**

该函数遍历每个顶点并打印其相邻的边，每个顶点的邻接边在所有顶点中总共被遍历一次。故整体时间复杂度为：**O(V+E)**。

**【DFSTraverse和DFS】—— 深度优先遍历**

在最坏情况下，每个顶点和每条边都会被访问一次。因此，**DFS**的时间复杂度是：**O(V+E)**。

**【BFSTraverse】—— 广度优先遍历**

和**DFS**类似，**BFS**也会最坏情况下遍历每个顶点和每条边一次。因此，BFS的时间复杂度也是：**O(V+E)**。

**综上所述，所有操作的总时间复杂度为：O(V+E)。**

**2.空间复杂度分析**

**【邻接表的存储】**

邻接表的空间复杂度主要由存储顶点和边的结构所决定。占用的空间为：**O(V+E)**。

【**Visited数组**】

保存已访问顶点状态的数组**visited**，其空间复杂度为：**O(V)**。

**【队列的空间】**

**BFS**使用的队列**SqQueue**最多可能需要存储所有顶点，因此其空间复杂度为：**O(V)**。

**综上所述，整体的空间复杂度为：O(V+E)。**

**七、实验总结**

1. 在**CreateUDG**中，输入的顶点数和边数等应进行有效性检查，以防止用户输入不合理的值（如负数或过大的值）。

2. 考虑验证输入的边**（v1, v2）**是否在已定义的顶点范围内，以避免在构建图时出现错误。

3. 对于**DFS**遍历，深度过大会导致**栈溢出**，尤其在图较大时。可以考虑使用**非递归方法实现DFS**。