**实验四：图的实现**

**学号：B21080526 班级：B210416 姓名：单家俊**

1. **实验目的**

1、在邻接矩阵或邻接表两种不同存储结构上实现图的基本运算的算法。设计主函数，测试上述算法。

2、实现图的深度优先或广度优先遍历。

1. **实验任务**

以邻接矩阵或邻接表定义无向图，其中每个元素的数据类型为int。实现图的深度优先或者广度优先遍历，打印所有的节点。

1. **实验设备**

Visual Studio

1. **实验内容（步骤）**

1、通过程序实现图的数据结构。

2、实现图的初始化和创建。

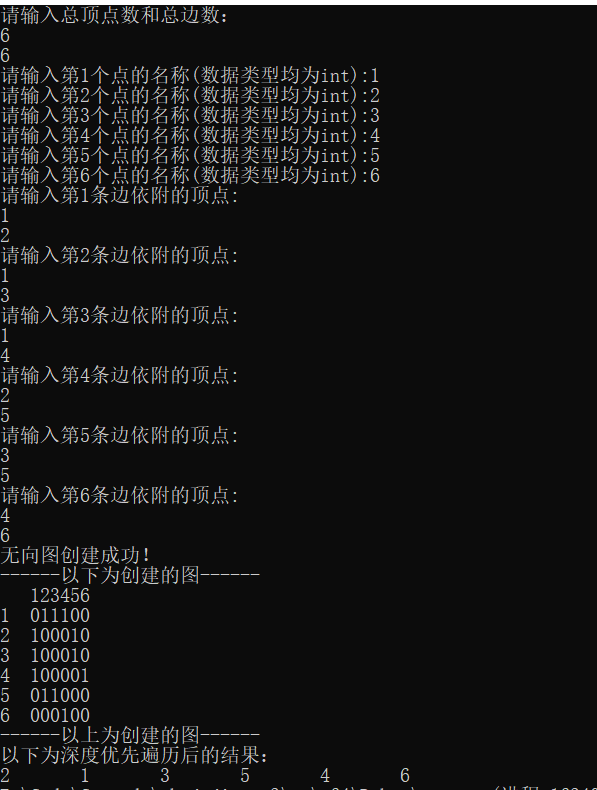
3、实现图的深度优先遍历，并将遍历结果打印出来。

1. **实验结果（总结、分析）**

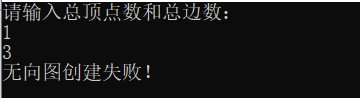
**实验结果分析：**

通过程序实现下图所示的图





程序运行之后，依照提示先后输入总顶点数、总边数，每个顶点的名称和每条边依附的顶点，创建完成后，程序将打印图的邻接矩阵，最后将调用DFS并将遍历节点按先后顺序打印出来。



如果输入的顶点数和边数不合法，将提示创建无向图失败。

**实验过程中遇到的问题：**

1、在深度优先遍历的过程中，对辅助数组visited[]的使用不太熟练。

2、对于递归结束的判断不太熟练。

3、长时间不回顾知识，遗忘的较快。

**体会：**

1、经过这次图的基本运算与遍历算法实验，让我掌握了图的两种表示方法：邻接矩阵表示和邻接表表示以及对应的代码实现，相关的数据结构，基本运算：初始化、插入、删除、搜索、撤销等等。

2、熟练掌握了深度优先遍历DFS和宽度优先遍历BFS的基本原理，以及算法实现，算法效率分析。

1. **附录（代码）**

#include <iostream>

using namespace std;

#define OK 1

#define ERROR 0

#define MVNum 100 //最大顶点数

typedef int Status;

typedef int VerTexType; //假设顶点的数据类型为int

typedef int ArcType; //假设边的权值类型为int

typedef struct {

VerTexType vexs[MVNum]; //顶点表

ArcType arcs[MVNum][MVNum]; //邻接矩阵

int vexnum, arcnum; //图的当前点数和边数

}AMGraph;

int LocateVex(AMGraph G, VerTexType u)

{//存在则返回u在顶点表中的下标;否则返回-1

int i;

for (i = 0; i < G.vexnum; ++i)

if (u == G.vexs[i]) //判断u是否在顶点表中

return i;

return -1;

}

Status CreateUDN(AMGraph &G) {

//采用邻接矩阵表示法，创建无向网G

int i,j,k;

VerTexType v1, v2;

cout << "请输入总顶点数和总边数：" << endl;

cin >> G.vexnum >> G.arcnum; //输入总顶点数，总边数

if (G.arcnum > (G.vexnum\*(G.vexnum-1))/2)

return ERROR;

for (i = 0; i < G.vexnum; ++i)

{

cout << "请输入第" << (i + 1) << "个点的名称(数据类型均为int):";

cin >> G.vexs[i];

}//依次输入点的信息，顶点表

for (i = 0; i < G.vexnum; ++i) //初始化邻接矩阵，边的权值均置为极大值

{

for (j = 0; j < G.vexnum; ++j)

G.arcs[i][j] = 0;

}

for (k = 0; k < G.arcnum; ++k)

{

//构造邻接矩阵

cout << "请输入第" << (k + 1) << "条边依附的顶点:" << endl;

cin >> v1 >> v2 ; //输入一条边依附的顶点及权值

i = LocateVex(G, v1);

j = LocateVex(G, v2); //确定v1和v2在顶点表中的位置

G.arcs[i][j] = 1; //边<v1, v2>的权值设置为w

G.arcs[j][i] = G.arcs[i][j]; //设置<v1, v2>的对称边<v2, v1>的权值为w

}

return OK;

}

//深度优先搜索

bool visited[MVNum];

void DFS(AMGraph G, int v)

{

cout << G.vexs[v-1]<<"\t";

visited[v] = true; //访问第v个顶点

for (int w = 1; w <= G.vexnum; w++) //依次检查邻接矩阵v所在的行

{

if ((G.arcs[v-1][w-1] != 0) && (!visited[w]))

DFS(G, w);

}

//w是v的邻接点，如果w未访问，则递归调用DFS

}

void PrintGraph(AMGraph G)

{

int i;

int j;

cout << "------以下为创建的图------" << endl;

cout << " ";

for (i = 0; i < G.arcnum; i++)

cout << G.vexs[i];

cout << endl;

for (i = 0; i < G.arcnum; i++)

{

cout << G.vexs[i]<<" ";

for (j = 0; j < G.arcnum; j++)

cout << G.arcs[i][j];

cout << endl;

}

cout << "------以上为创建的图------" << endl;

}

int main()

{

int x;

AMGraph G;

x = CreateUDN(G);

if (x == 1)

{

cout << "无向图创建成功！" << endl;

PrintGraph(G);

cout << "以下为深度优先遍历后的结果：" << endl;

DFS(G,2);

}

else

{

cout << "无向图创建失败！" << endl;

}

return 0;

}