计算机与信息工程学院实验报告

••••••••••••••••••••••••••••••••• 密 ••••••••••••••••••••••••••••••••• 封 ••••••••••••••••••••••••••••••••• 线 •••••••••••••••••••••••••••••••••

姓名：\_杨佳森 学号：\_2112080106\_专业：\_数据科学与大数据技术年级：\_2021\_

课程： 数据结构 主讲教师：\_刘成\_\_ 辅导教师：\_\_ 无\_

实验时间：\_2023\_\_年 \_5\_月 \_9日 下午\_17\_时至19\_时，实验地点\_计算机大楼213\_\_

实验题目： 构造无向图（算法6.1），深度和广度遍历

实验目的： 通过该实验，使学生掌握图的几种存储结构，理解图的深度优先和广度优先遍历算法的思想和实现办法。

实验环境（硬件和软件） PC和vs code

实验内容：

实现教材算法7.2利用邻接矩阵构造无向图的算法，提供从邻接矩阵获得邻接表的功能，在此基础上进行深度优先遍历和广度优先遍历。

实验步骤：

1、初始化队列

void InitQueue(LinkQueue &Q) {

    QueuePtr q = new QNode();

    Q.front = q;

    if(Q.front == 0) {

        cout << "初始化失败！" << endl;

        return;

    }

    Q.rear = Q.front;

    Q.front->next = 0;

}

2、队列插入元素

void EnQueue(LinkQueue &Q,int p) {

    if(Q.front == 0) {

        cout << "请先初始化！" << endl;

        return;

    }

    QueuePtr q = new QNode();

    q->next = 0;

    q->data = p;

    Q.rear->next = q;

    Q.rear = q;

}

3、出队函数

void DeQueue(LinkQueue &Q,int &p) {

    if(Q.front == 0) {

        cout << "请先初始化！" << endl;

        return;

    }

    if(Q.front == Q.rear) {

        cout << "该图为空！" << endl;

        return;

    }

    QueuePtr q;

    q = Q.front->next;//让q指向队列的首元结点

    p = q->data;

    Q.front->next = q->next;

    if(q == Q.rear) Q.rear = Q.front;//保留尾指针，防止尾指针被删除

    delete q;

}

4、定位函数（返回点在数组中的位置）

int LocateVex(AMGraph G,int g) {

    int i;

    for(i = 0; i < G.vexnum; i++) {

        if(g == G.vex[i]) return i;//将顶点在数组中的位置返回回来

    }

    return overflow;

}

5、创建无向图

void CreatUDN(AMGraph &G) {

    int i,j,v1,v2,k;

    cout << "请输入图的顶点数和边数：" << endl;

    cin >> G.vexnum >> G.arcnum;

    cout << "为顶点数组赋值：" << endl;

    for(i = 0; i < G.vexnum; i++) cin>>G.vex[i];//为顶点数组赋值  0对应1

    for(i = 0; i < G.vexnum; i++) {

        for(j = 0; j < G.vexnum; j++) G.arcs[i][j] = 0;

    }//先将矩阵中全部元素赋值为0

    for(k = 0; k < G.arcnum; k++) {

    cout << "输入有关联的两个顶点：" << endl;

    cin >> v1 >> v2;

    i = LocateVex(G,v1);

    j = LocateVex(G,v2);

    G.arcs[i][j] = 1;

    G.arcs[j][i] = G.arcs[i][j];

    }

    cout << "无向图构建完毕！" << endl;

}

6、输出邻接矩阵

void Printf(AMGraph G) {

    int i,j;

    for(i = 0; i < G.vexnum; i++) {

        for(j = 0; j < G.vexnum; j++)

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

        cout << endl;

    }

}

7、深度优先遍历

void DFS(AMGraph G,int w) {

    int i,j;

    cout << G.vex[w] << " ";

    Visited[w] = 1;

    for(j = 0; j < G.vexnum; j++) {

        if((G.arcs[w][j] == 1) && (Visited[j] != 1))

            DFS(G,j);

    }

}

8、广度优先遍历

void BFS(AMGraph G,int v) {

    int i;

    for(i = 0; i < G.vexnum; i++)

        Visited[i] = 0;

    for(v = 0; v < G.vexnum; v++) {

        if(Visited[v] == 0) {

            int u,w,temp,temp1,temp2;

            cout << G.vex[v] << " ";

            Visited[v] = 1;

            LinkQueue Q;

            InitQueue(Q);

            temp = G.vex[v];

            EnQueue(Q,temp);

            while(Q.front != Q.rear) {

                DeQueue(Q,u);

                temp1 = u - 1;

                for(w = FirstAdjVex(G,temp1); w >= 0; w = NextAdjVex(G,temp1,w)) {

                    if(Visited[w] == 0) {

                        cout << G.vex[w] << " ";

                        Visited[w] = 1;

                        temp2 = G.vex[w];

                        EnQueue(Q,temp2);

                    }

                }

            }

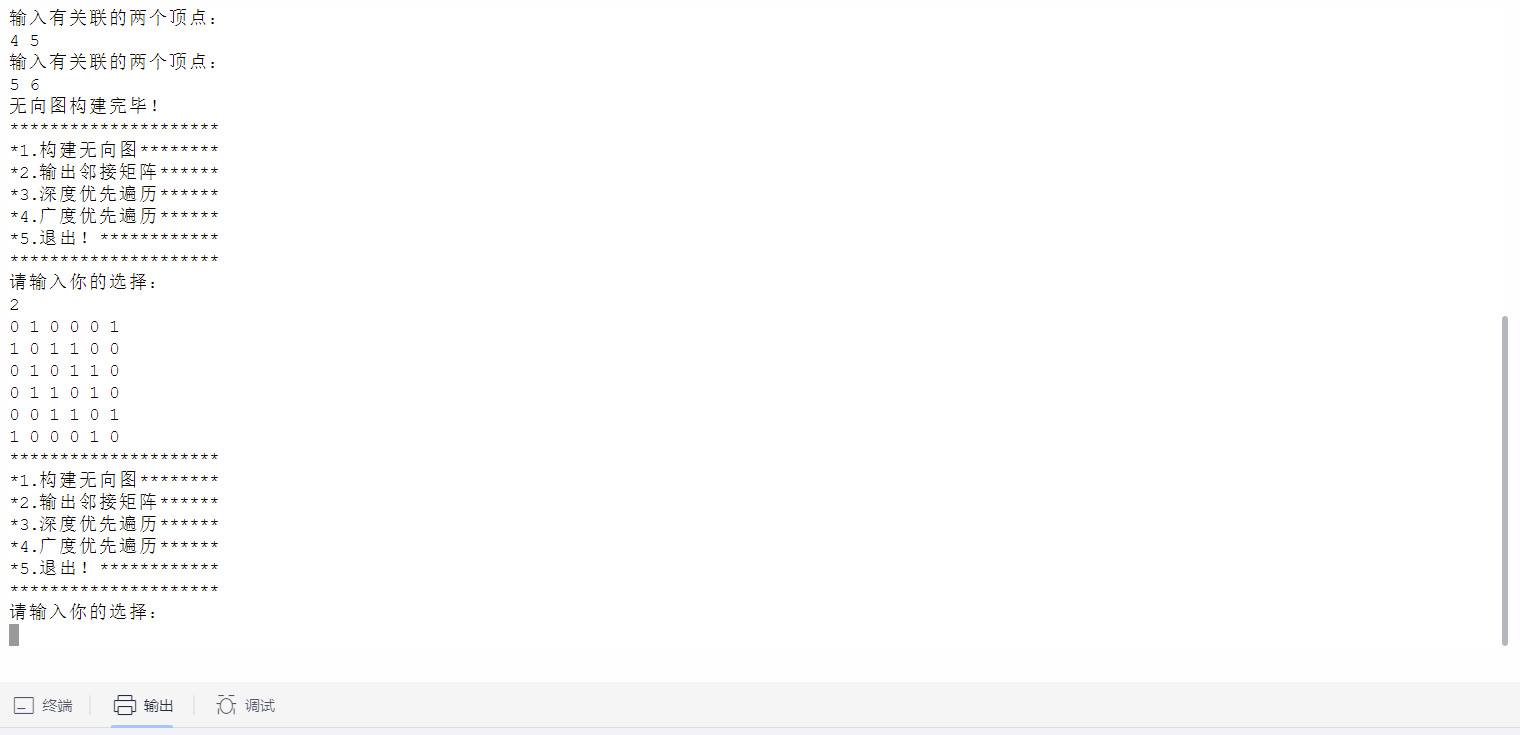
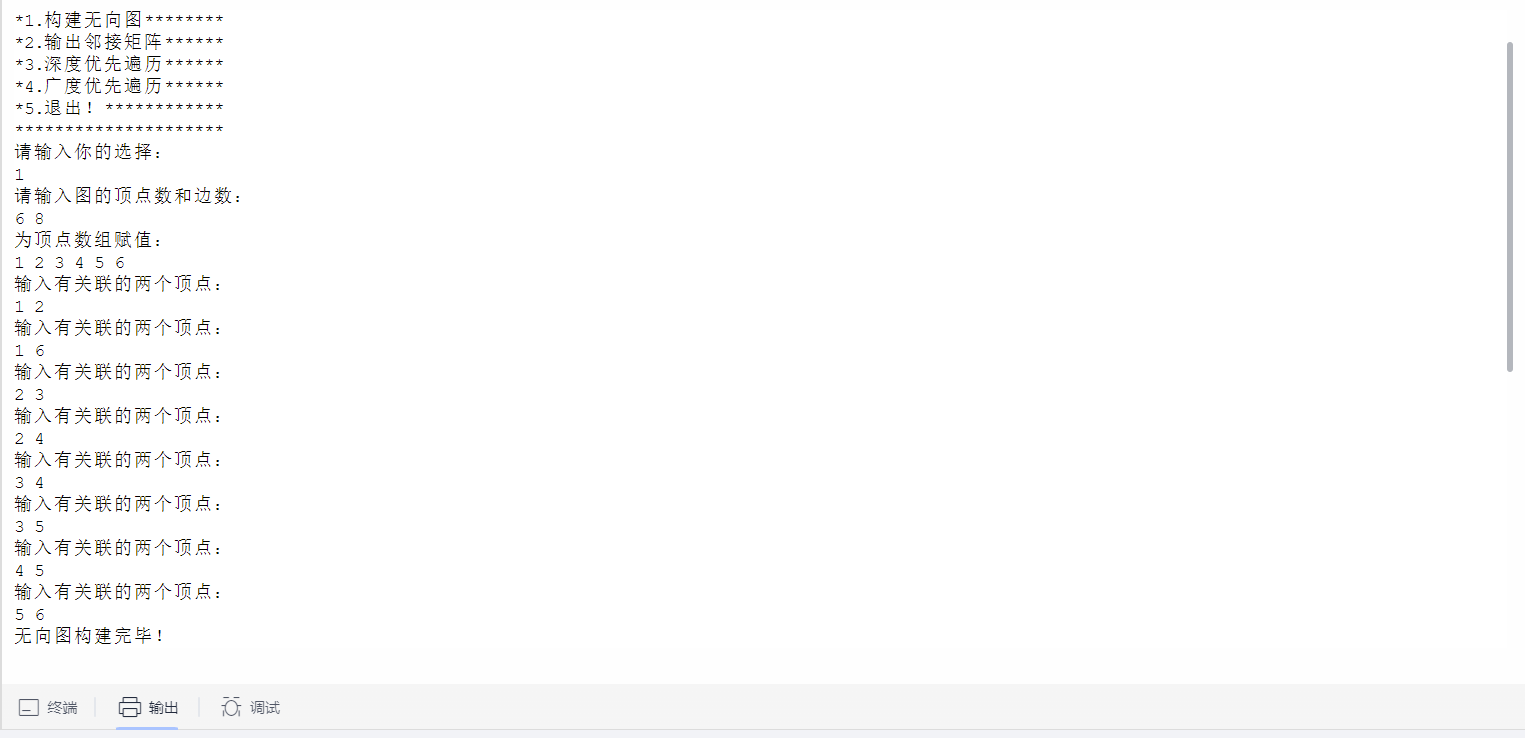
        }

    }

}

实验数据记录：

1、创建无向图，屏幕输出邻接矩阵



2、深度优先遍历，屏幕输出1 2 3 4 5 6



3、广度优先遍历，屏幕输出1 2 6 3 4 5



问题讨论：

深度优先遍历：对每一个可能的分支路径深入到不能再深入为止，而且每个结点只能访问一次。

广度优先遍历：又叫层次遍历，从上往下对每一层依次访问，在每一层中，从左往右（也可以从右往左）访问结点，访问完一层就进入下一层，直到没有结点可以访问为止。

源代码：

#include <iostream>

using namespace std;

#define maxsize 100

#define overflow -1

int Visited[maxsize] = {0};

typedef struct {

    int vex[maxsize];

    int arcs[maxsize][maxsize];

    int vexnum,arcnum;

}AMGraph;

typedef struct QNode {

    int data;

    struct QNode \*next;

}QNode,\*QueuePtr;

typedef struct {

    QueuePtr front;

    QueuePtr rear;

}LinkQueue;

//初始化队列

void InitQueue(LinkQueue &Q) {

    QueuePtr q = new QNode();

    Q.front = q;

    if(Q.front == 0) {

        cout << "初始化失败！" << endl;

        return;

    }

    Q.rear = Q.front;

    Q.front->next = 0;

}

//队列插入元素

void EnQueue(LinkQueue &Q,int p) {

    if(Q.front == 0) {

        cout << "请先初始化！" << endl;

        return;

    }

    QueuePtr q = new QNode();

    q->next = 0;

    q->data = p;

    Q.rear->next = q;

    Q.rear = q;

}

//出队函数

void DeQueue(LinkQueue &Q,int &p) {

    if(Q.front == 0) {

        cout << "请先初始化！" << endl;

        return;

    }

    if(Q.front == Q.rear) {

        cout << "该图为空！" << endl;

        return;

    }

    QueuePtr q;

    q = Q.front->next;//让q指向队列的首元结点

    p = q->data;

    Q.front->next = q->next;

    if(q == Q.rear) Q.rear = Q.front;//保留尾指针，防止尾指针被删除

    delete q;

}

//定位函数

int LocateVex(AMGraph G,int g) {

    int i;

    for(i = 0; i < G.vexnum; i++) {

        if(g == G.vex[i]) return i;//将顶点在数组中的位置返回回来

    }

    return overflow;

}

//创建无向图

void CreatUDN(AMGraph &G) {

    int i,j,v1,v2,k;

    cout << "请输入图的顶点数和边数：" << endl;

    cin >> G.vexnum >> G.arcnum;

    cout << "为顶点数组赋值：" << endl;

    for(i = 0; i < G.vexnum; i++) cin>>G.vex[i];//为顶点数组赋值  0对应1

    for(i = 0; i < G.vexnum; i++) {

        for(j = 0; j < G.vexnum; j++) G.arcs[i][j] = 0;

    }//先将矩阵中全部元素赋值为0

    for(k = 0; k < G.arcnum; k++) {

    cout << "输入有关联的两个顶点：" << endl;

    cin >> v1 >> v2;

    i = LocateVex(G,v1);

    j = LocateVex(G,v2);

    G.arcs[i][j] = 1;

    G.arcs[j][i] = G.arcs[i][j];

    }

    cout << "无向图构建完毕！" << endl;

}

//输出邻接矩阵

void Printf(AMGraph G) {

    int i,j;

    for(i = 0; i < G.vexnum; i++) {

        for(j = 0; j < G.vexnum; j++)

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

        cout << endl;

    }

}

//深度优先遍历

void DFS(AMGraph G,int w) {

    int i,j;

    cout << G.vex[w] << " ";

    Visited[w] = 1;

    for(j = 0; j < G.vexnum; j++) {

        if((G.arcs[w][j] == 1) && (Visited[j] != 1))

            DFS(G,j);

    }

}

//调用DFS函数

void DFSTraverse(AMGraph G,int v) {

    int i;

    for(i = 0; i < G.vexnum; i++)

        Visited[i] = 0;

    for(v = 0; v < G.vexnum; v++) {

        if(Visited[v] == 0)

            DFS(G,v);

    }

}

//定位函数1

int FirstAdjVex(AMGraph G,int u) {

    int i;

    for(i = 0; i < G.vexnum; i++) {

        if(G.arcs[u][i] == 1)

            return i;

    }

    return overflow;

}

//定位函数2

int NextAdjVex(AMGraph G,int u,int w) {

    int i;

    for(i = w + 1; i < G.vexnum; i++) {

        if(G.arcs[u][i] == 1)

            return i;

    }

    return overflow;

}

//广度优先遍历

void BFS(AMGraph G,int v) {

    int i;

    for(i = 0; i < G.vexnum; i++)

        Visited[i] = 0;

    for(v = 0; v < G.vexnum; v++) {

        if(Visited[v] == 0) {

            int u,w,temp,temp1,temp2;

            cout << G.vex[v] << " ";

            Visited[v] = 1;

            LinkQueue Q;

            InitQueue(Q);

            temp = G.vex[v];

            EnQueue(Q,temp);

            while(Q.front != Q.rear) {

                DeQueue(Q,u);

                temp1 = u - 1;

                for(w = FirstAdjVex(G,temp1); w >= 0; w = NextAdjVex(G,temp1,w)) {

                    if(Visited[w] == 0) {

                        cout << G.vex[w] << " ";

                        Visited[w] = 1;

                        temp2 = G.vex[w];

                        EnQueue(Q,temp2);

                    }

                }

            }

        }

    }

}

void show() {

    cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

    cout << "\*1.构建无向图\*\*\*\*\*\*\*\*" << endl;

    cout << "\*2.输出邻接矩阵\*\*\*\*\*\*" << endl;

    cout << "\*3.深度优先遍历\*\*\*\*\*\*" << endl;

    cout << "\*4.广度优先遍历\*\*\*\*\*\*" << endl;

    cout << "\*5.退出！\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

    cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

}

int main() {

    int i,j,w;

    int choice;

    AMGraph G;

    LinkQueue Q;

    while(true) {

        show();

        cout << "请输入你的选择：" << endl;

        cin >> choice;

        switch(choice) {

            case 1:

                CreatUDN(G);

                break;

            case 2:

                Printf(G);

                break;

            case 3:

                cout << "请输入你想要遍历的起点：" << endl;

                cin >> w;

                cout << "深度优先遍历为：" << endl;

                DFSTraverse(G,w);

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

                    Visited[i] = 0;

                cout << endl;

                break;

            case 4:

                cout << "请输入你想要遍历的起点：" << endl;

                cin >> w;

                BFS(G,w);

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

                    Visited[i] = 0;

                cout << endl;

                break;

            case 5:

                exit(0);

                break;

            default :

                cout << "非法输入！" << endl;

                break;

        }

    }

    return 0;

}