**实验3**

**实验题目：图的遍历操作**

**实验目的：**

1）进一步熟悉利用Code::Blocks，VC++等集成编译环境调试C/C++代码；

2）熟练掌握有向图和无向图的概念；

3）熟练掌握邻接矩阵和邻接链表建立图的存储结构；

4）熟练掌握图的深度优先遍历（DFS）与广度优先遍历算法（BFS）的实现。

**实验要求：**

阅读所提供的代码，根据提示补全缺少的部分，完成下面的功能：

采用邻接矩阵和邻接链表作为图的存储结构，完成有向图和无向图的DFS和BFS操作。

**实验主要步骤：**

1. 分析、理解给出的程序，根据提示填写缺少部分的代码；
2. 调试程序，并且编译通过；
3. 实现功能：采用邻接矩阵和邻接链表，要求输入的内容如下：

Input VertexNum(n) and EdgesNum(e): 8,9

Input Vertex string: 01234567

Input edges,Creat Adjacency Matrix

0 1

0 2

1 3

1 4

2 5

2 6

3 7

4 7

5 6

然后输出该图的DFS（深度优先遍历）和BFS（广度优先遍历）的结果。

4、完成思考题

**课后实验报告提交要求：**

1. 使用邻接矩阵的实现与使用邻接表的实现分别写在一个源文件(.c文件或者.cpp文件)中（即，一共两个源文件）；
2. 实验报告打印纸质版提交（11月27日）。

程序代码（请同学们填写 **程序1 以及 程序2** 空缺部分的代码）:

**程序1：邻接矩阵作为存储结构**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include"stdio.h"

#include"stdlib.h"

#define MaxVertexNum 100 //定义最大顶点数

typedef struct {

char vexs[MaxVertexNum]; //顶点表

int edges[MaxVertexNum][MaxVertexNum]; //邻接矩阵

int n, e; //图中的顶点数n和边数e

}MGraph;

//=========建立邻接矩阵=======

void CreatMGraph(MGraph& G)

{

int i, j, k;

char a;

printf("Input VertexNum(n) and EdgesNum(e): ");

scanf("%d,%d", &G.n, &G.e); //输入顶点数和边数

scanf("%c", &a);

printf("Input Vertex string:");

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

{

scanf("%c", &a);

G.vexs[i] = a; //读入顶点信息，建立顶点表

}

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

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

G.edges[i][j] = 0; //初始化邻接矩阵

printf("Input edges,Creat Adjacency Matrix\n");

for (k = 0; k < G.e; k++) { //读入e条边，建立邻接矩阵

scanf("%d%d", &i, &j); //输入边（Vi，Vj）的顶点序号

G.edges[i][j] = 1;

G.edges[j][i] = 1;

}

}

//=========定义标志向量，为全局变量=======

typedef enum { FALSE, TRUE } Boolean;

Boolean visited[MaxVertexNum];

//========DFS：深度优先遍历的递归算法======

void DFSM(MGraph& G, int i)

{

printf("%c", G.vexs[i]); // 访问顶点Vi

visited[i] = TRUE; // 将顶点Vi标记为已访问

for (int j = 0; j < G.n; j++) {

if (G.edges[i][j] == 1 && !visited[j]) {

DFSM(G, j); // 递归访问与Vi相邻且未访问过的顶点

}

}

}

void DFS(MGraph& G)

{

int i;

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

visited[i] = FALSE; //标志向量初始化

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

if (!visited[i]) //Vi未访问过

DFSM(G, i); //以Vi为源点开始DFS搜索

}

//===========BFS：广度优先遍历=======

void BFS(MGraph& G, int k)

{ //以Vk为源点对用邻接矩阵表示的图G进行广度优先搜索

int i, j, f = 0, r = 0;

int cq[MaxVertexNum]; //定义队列

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

visited[i] = FALSE; //标志向量初始化

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

cq[i] = -1; //队列初始化

printf("%c", G.vexs[k]); //访问源点Vk

visited[k] = TRUE;

cq[r] = k; //Vk已访问，将其入队。

while (cq[f] != -1 && f < G.n) {

i = cq[f]; f = f + 1;

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

{

if (G.edges[i][j] == 1 && !visited[j]) {

visited[j] = TRUE;

r = r + 1;

cq[r] = j;

printf("%c", G.vexs[j]);

}

}

}

}

//==========main=====

int main()

{

MGraph G;

CreatMGraph(G); //建立邻接矩阵

printf("Print Graph DFS: ");

DFS(G); //深度优先遍历

printf("\n");

printf("Print Graph BFS: ");

BFS(G, 3); //以序号为3的顶点开始广度优先遍历

printf("\n");

}

**程序2：邻接链表作为存储结构**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include"stdio.h"

#include"stdlib.h"

#define MaxVertexNum 50 //定义最大顶点数

typedef struct node { //边表结点

int adjvex; //邻接点域

struct node\* next; //链域

}EdgeNode;

typedef struct vnode { //顶点表结点

char vertex; //顶点域

EdgeNode\* firstedge; //边表头指针

}VertexNode;

typedef VertexNode AdjList[MaxVertexNum]; //AdjList是邻接表类型

typedef struct {

AdjList adjlist; //邻接表

int n, e; //图中当前顶点数和边数

} ALGraph; //图类型

//=========建立图的邻接表=======

void CreatALGraph(ALGraph& G)

{

int i, j, k;

char a;

EdgeNode\* s; //定义边表结点

printf("Input VertexNum(n) and EdgesNum(e): ");

scanf("%d,%d", &G.n, &G.e); //读入顶点数和边数

scanf("%c", &a);//读取换行符

printf("Input Vertex string:");

//建立边表

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

{

scanf("%c", &G.adjlist[i].vertex); // 读入顶点信息

G.adjlist[i].firstedge = NULL; // 边表头指针初始化为NULL

}

scanf("%c", &a); // 读取换行符

printf("Input edges,Creat Adjacency List\n");

for (k = 0; k < G.e; k++) {

scanf("%d %d", &i, &j); //读入边（Vi，Vj）的顶点对序号

s = (EdgeNode\*)malloc(sizeof(EdgeNode)); //生成边表结点

s->adjvex = j; //邻接点序号为j

s->next = G.adjlist[i].firstedge;

G.adjlist[i].firstedge = s; //将新结点\*S插入顶点Vi的边表头部

s = (EdgeNode\*)malloc(sizeof(EdgeNode));

s->adjvex = i; //邻接点序号为i

s->next = G.adjlist[j].firstedge;

G.adjlist[j].firstedge = s;

}

}

//=========定义标志向量，为全局变量=======

typedef enum { FALSE, TRUE } Boolean;

Boolean visited[MaxVertexNum];

//========DFS：深度优先遍历的递归算法======

void DFSM(ALGraph& G, int i)

{ //以Vi为出发点对邻接链表表示的图G进行DFS搜索

EdgeNode\* p;

printf("%c", G.adjlist[i].vertex); //访问顶点Vi

visited[i] = TRUE; //标记Vi已访问

p = G.adjlist[i].firstedge; //取Vi边表的头指针

while (p) { //依次搜索Vi的邻接点Vj，这里j=p->adjvex

if (!visited[p->adjvex]) {

DFSM(G, p->adjvex); //以Vj为源点进行深度优先遍历

}

p = p->next;

}

}

void DFS(ALGraph& G)

{

int i;

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

visited[i] = FALSE; //标志向量初始化

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

if (!visited[i]) //Vi未访问过

DFSM(G, i); //以Vi为源点开始DFS搜索

}

//==========BFS：广度优先遍历=========

void BFS(ALGraph& G, int k) { //以Vk为源点对用邻接链表表示的图G进行广度优先搜索

int i, f = 0, r = 0; EdgeNode\* p; int cq[MaxVertexNum]; //定义FIFO队列

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

visited[i] = FALSE; //标志向量初始化

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

cq[i] = -1; //初始化标志向量

printf("%c", G.adjlist[k].vertex); //访问源点Vk

visited[k] = TRUE;

cq[r] = k; //Vk已访问，将其入队。注意，实际上是将其序号入队

while (cq[f] != -1) { //队列非空则执行

i = cq[f]; f = f + 1; //Vi出队

p = G.adjlist[i].firstedge; //取Vi的边表头指针

while (p) { //依次搜索Vi的邻接点Vj（令p->adjvex=j）

if (!visited[p->adjvex]) {

printf("%c", G.adjlist[p->adjvex].vertex); //访问顶点Vj

visited[p->adjvex] = TRUE;

cq[r] = p->adjvex; //将顶点Vj入队

r++;

}

p = p->next;

}

}

}//endwhile

//==========主函数===========

int main()

{

ALGraph G;

CreatALGraph(G);

printf("Print Graph DFS: ");

DFS(G);

printf("\n");

printf("Print Graph BFS: ");

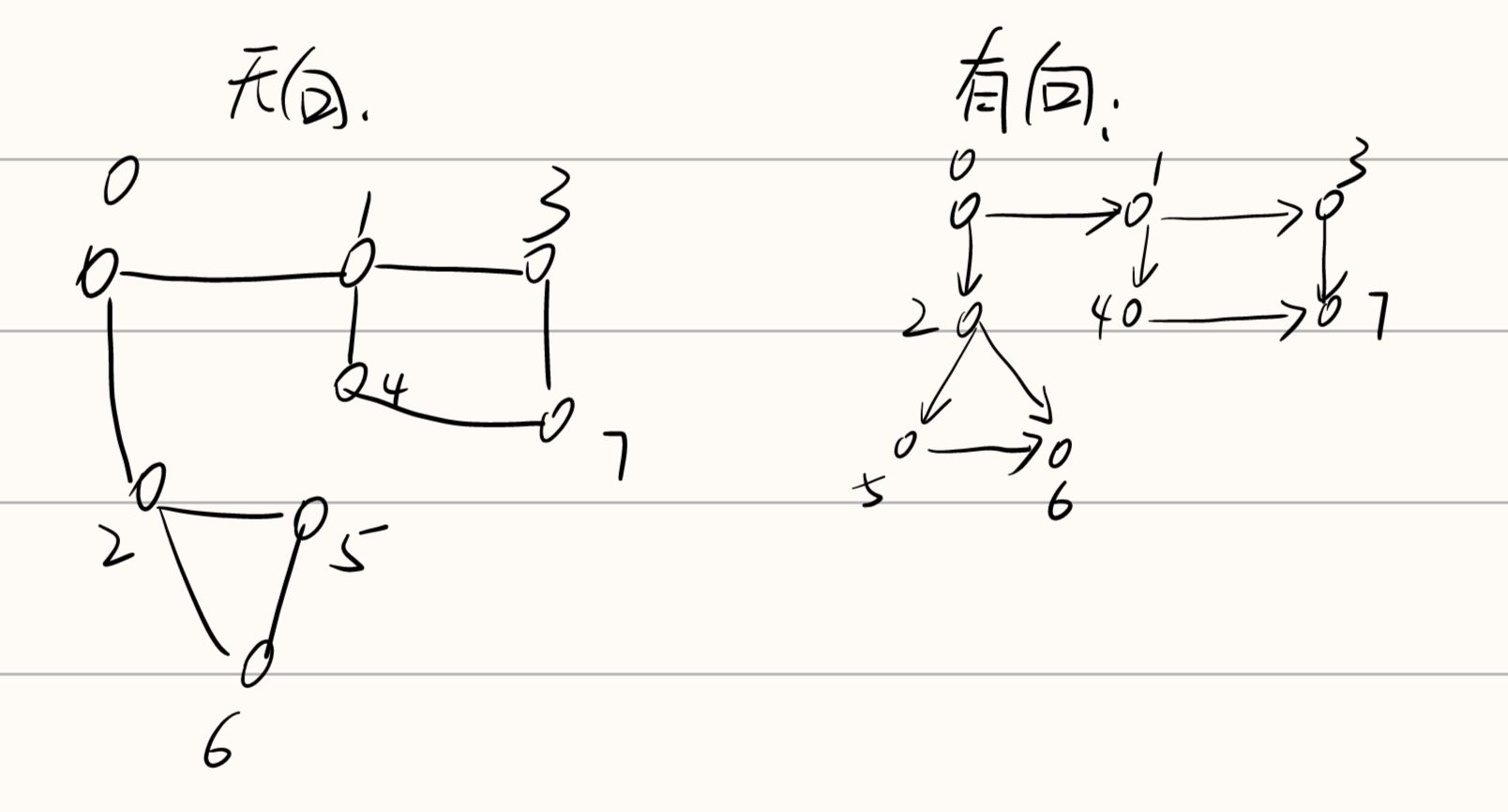
BFS(G, 3);

printf("\n");

}

**思考题：**

1. 根据实验步骤中提供的顶点与边分别画出对应无向图和有向图的示意图。



1. 若要将**程序1和程序2**应用于有向图的遍历，则需要分别对代码做什么修改？

程序1：

在CreatMGraph函数中，修改邻接矩阵的建立方式，有向图的邻接矩阵中，表示边的位置只需设置为1。在DFSM函数与BFS函数中，将对邻接矩阵中边的判断改为G.edges[i][j] == 1，表示从顶点Vi到顶点Vj存在有向边。

程序2：

在CreatALGraph中，判断数字在前则由该数字出发，对该数字来说可加tag=1，可以通过tag值判断方向。在函数DFSM和BFS中，对遍历过程中访问边的方式进行修改，需要考虑边的方向信息，只访问符合遍历顺序的边。可以添加条件if (tag == 1)来判断边的方向是否为正向边，只有符合条件的边才进行递归遍历。

1. 分析程序1和程序2中DFS和BFS（顶点数为n，边数为e）的时间复杂度，并指出决定算法时间复杂度的主要代码段。

程序1：

DFS中为邻接矩阵建立与递归：O(n^2)，建立时需要遍历所有的节点组合，递归中为O(n^2)，需要遍历所有的节点和边。故DFS的总时间复杂度为O(n^2)

BFS中邻接矩阵的建立与循环：O(n^2)，需要遍历所有的节点组合。

程序2

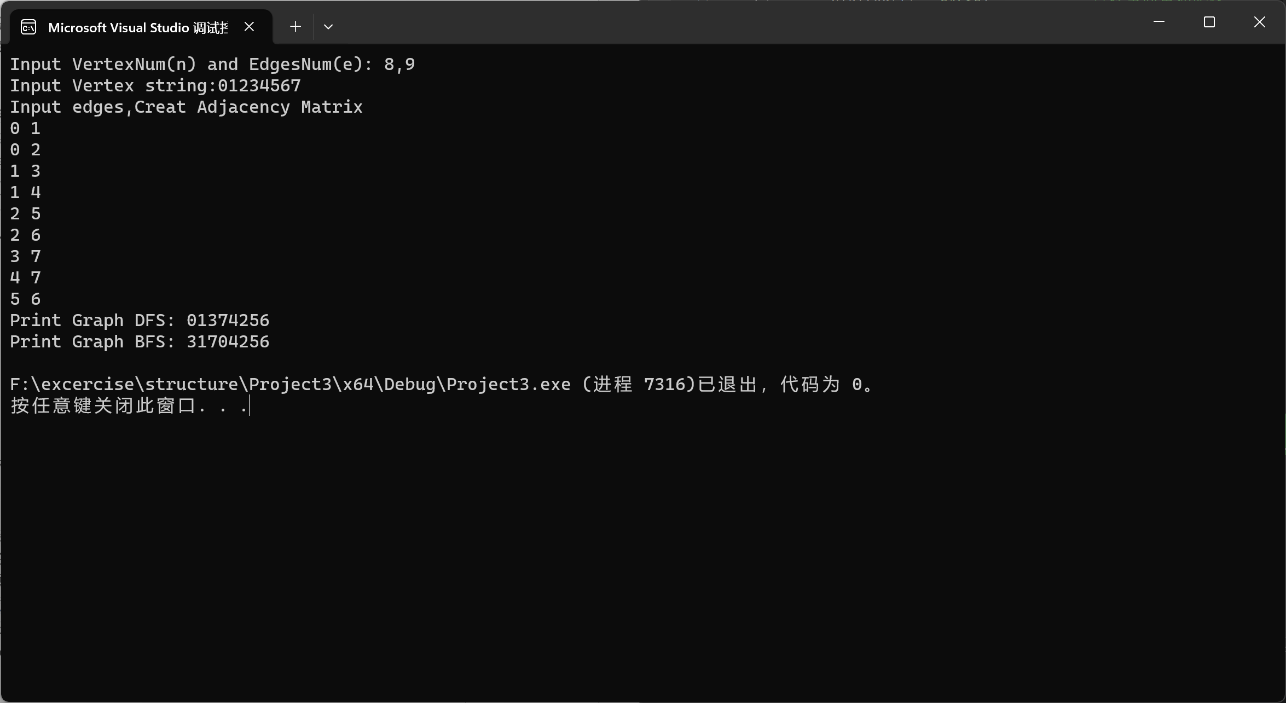
DFS函数中为递归部分：初始化遍历时候为O(e)，而递归时候为O（n+e）,所以总时间复杂度为O(n+e);故递归部分决定

BFS函数中为循环部分：O(n+e)，其需遍历所有结点与边

**实验结果：**

根据实验步骤中的提示，将屏幕输出内容截取，并粘贴到word文档中。

程序1



程序2

