# 

# 《计算机实践》

# 实验报告I—数据结构

班号：0321501 学号：032150115

姓名：易博天

Email：yiboxiaotian@nuaa.edu.cn

签名：

南京航空航天大学

## 实验：树和图的实验

### 一、实验目的

　　掌握二叉树的创建、遍历等操作实现；掌握图的邻接矩阵和邻接表创建及遍历等操作实现。。

### 二、实验内容

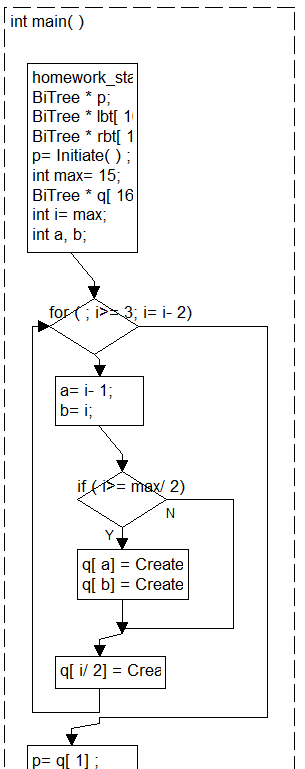
二叉树的创建、遍历及相关操作实现。

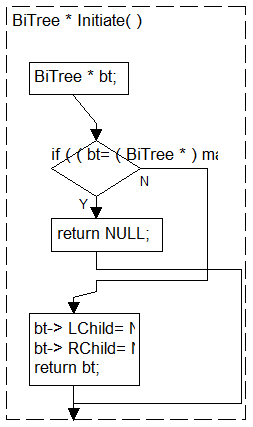
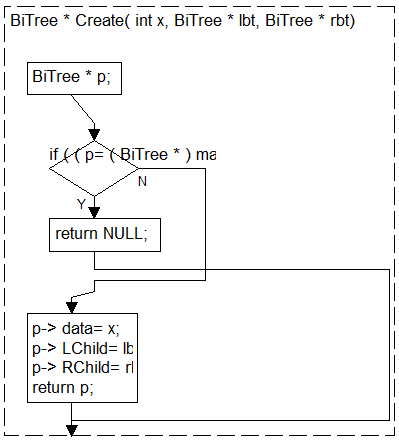
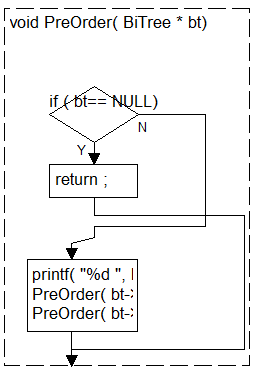
### 三、算法分析与程序设计

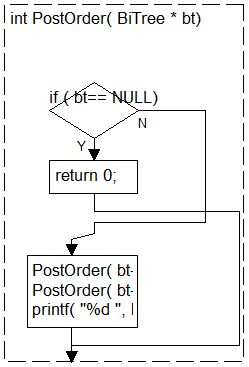
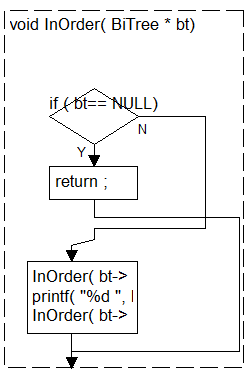
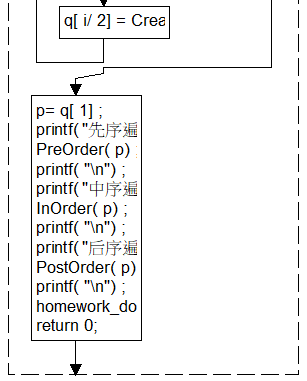
1. 编程实现二叉排序树的链式存储结构创建，并实现先序、中序、后序遍历（递归和非递归方法自选）操作，最后统计叶子结点的数目。

2.输入数据：自选方法输入结点个数及结点值，结点值使用不重复的正整数表示。 输出形式：输出正确的先序、中序、后序遍历序列，给出叶子结点数目。

3.算法描述

给出算法描述的程序流程图





4. 程序设计

给出实现程序功能的函数及说明如下：

(1) 本程序包含的函数

主函数 main()

BiTree \*Initiate(); // 建立一棵空二叉树

BiTree \*Create(ElemType x, BiTree \*lbt, BiTree \*rbt); // 建立一棵以x为根结点的数据域信息

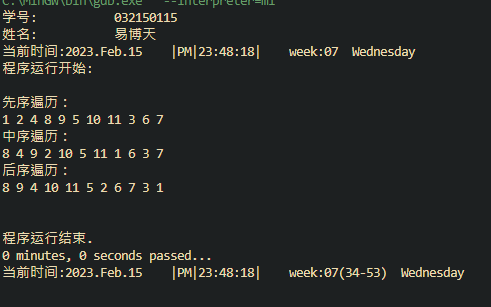
void PreOrder(BiTree \*bt); // 先序遍历二叉树bt

void InOrder(BiTree \*bt); // 中序遍历二叉树bt

int PostOrder(BiTree \*bt); // 后序遍历二叉树bt

### 四、调试分析

#### 1、调试过程及说明

* 以截屏图像的方式给出，包括程序运行开始和结束时的系统时间、姓名、学号、实验名称等信息.

#### 2、调试分析

照着书上写的，没什么问题，但是二叉树是给定的，也可以修改一下改为可输入的

### 五、使用说明

运行即可，二叉树是给定的

**六、测试结果**

内容：给出测试用例测试程序。写出测试的情况，使用的输入以及得到的结果。

要求：测试的情况越全面越好。

示例：

测试例子如上图所示

**七、实验的收获和感想**

照着书上写就行，先序中序后序的代码都差不多，毕竟只是换了个顺序

**八、附录：源代码（双栏排版，加必要的注释）**

#include "homework.h"

#define MaxSize 100

#define ElemType int

typedef struct bitree

{

ElemType data;

struct bitree \*LChild, \*RChild;

} BiTree;

BiTree \*Initiate(); // 建立一棵空二叉树

BiTree \*Create(ElemType x, BiTree \*lbt, BiTree \*rbt); // 建立一棵以x为根结点的数据域信息

void PreOrder(BiTree \*bt); // 先序遍历二叉树bt

void InOrder(BiTree \*bt); // 中序遍历二叉树bt

int PostOrder(BiTree \*bt); // 后序遍历二叉树bt

BiTree \*Initiate() // 建立一棵空二叉树

{

BiTree \*bt; // 初始化建立二叉树 \* bt的头结点

if ((bt = (BiTree \*)malloc(sizeof(BiTree))) == NULL)

return NULL;

bt->LChild = NULL;

bt->RChild = NULL;

return bt;

}

BiTree \*Create(ElemType x, BiTree \*lbt, BiTree \*rbt)

// 建立一棵以x为根结点的数据域信息

{

BiTree \*p;

if ((p = (BiTree \*)malloc(sizeof(BiTree))) == NULL)

return NULL;

p->data = x;

p->LChild = lbt;

p->RChild = rbt;

return p;

}

void PreOrder(BiTree \*bt)

{

// 先序遍历二叉树bt

if (bt == NULL)

return; // 递归调用的结束条件

printf("%d ", bt->data); // 访问结点的数据域

PreOrder(bt->LChild); // 先序递归遍历bt的左子树

PreOrder(bt->RChild); // 先序递归遍历bt的右子树

}

void InOrder(BiTree \*bt)

{

// 中序遍历二叉树bt

if (bt == NULL)

return; // 递归调用的结束条件

InOrder(bt->LChild); // 中序递归遍历bt的左子树

printf("%d ", bt->data); // 访问结点的数据域

InOrder(bt->RChild); // 中序递归遍历bt的右子树

}

int PostOrder(BiTree \*bt)

{

// 后序遍历二叉树bt

if (bt == NULL)

return 0; // 递归调用的结束条件

PostOrder(bt->LChild); // 后序递归遍历bt的左子树

PostOrder(bt->RChild); // 后序递归遍历bt的右子树

printf("%d ", bt->data); // 访问结点的数据域

}

int main()

{

homework\_start();

BiTree \*p;

BiTree \*lbt[16] = {NULL}; // 左子树指针数组

BiTree \*rbt[16] = {NULL}; // 右子树指针数组

p = Initiate();

int max = 15; // 满足max=2max+1

BiTree \*q[16]; // 头结点指针数组

int i = max;

int a, b;

for (; i >= 3; i = i - 2) // 循环建立特定二叉树

{

a = i - 1;

b = i;

if (i >= max / 2)

{

q[a] = Create(a, lbt[a], rbt[a]);

q[b] = Create(b, lbt[b], rbt[b]);

}

q[i / 2] = Create((i / 2), q[a], q[b]);

}

p = q[1];

printf("先序遍历：\n");

PreOrder(p);

printf("\n");

printf("中序遍历：\n");

InOrder(p);

printf("\n");

printf("后序遍历：\n");

PostOrder(p);

printf("\n");

homework\_done();

return 0;

}

# 实验 3.1.2 图的基本实验

# 二、实验内容

1.图的创建、遍历操作实现。

使用邻接矩阵存储图，实现深度优先遍历操作；使用邻接表存储图，实现广度优先遍

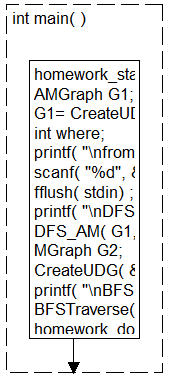
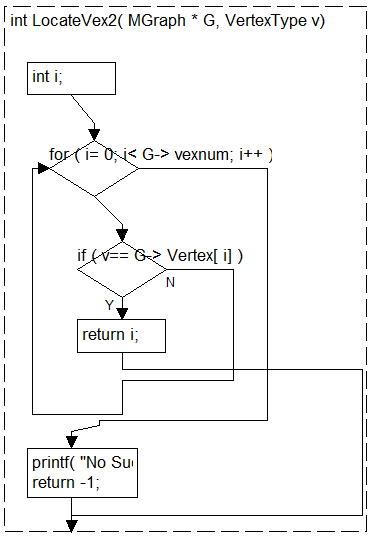
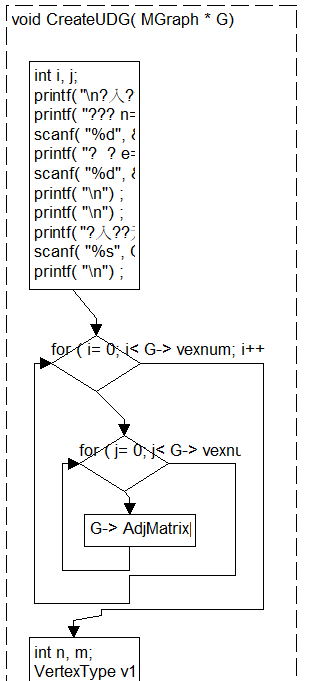
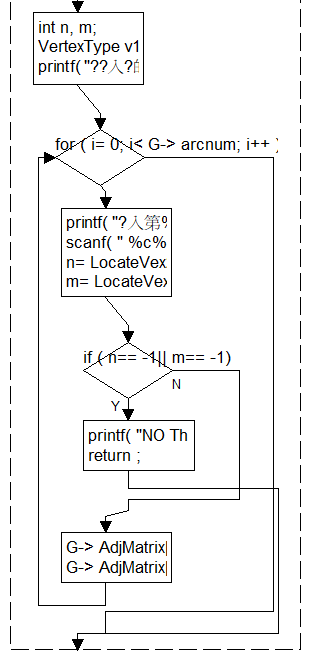
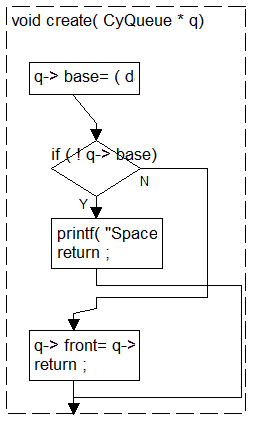
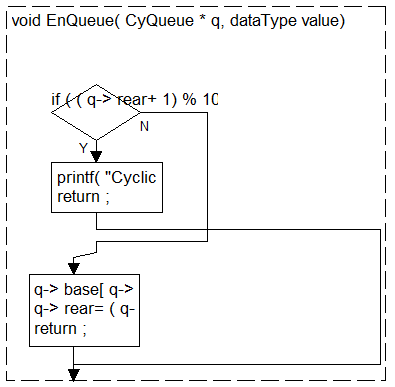
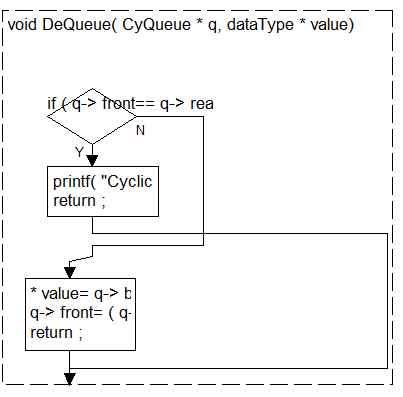
历操作。

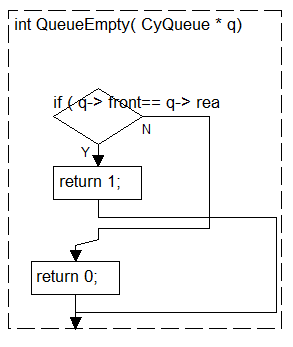
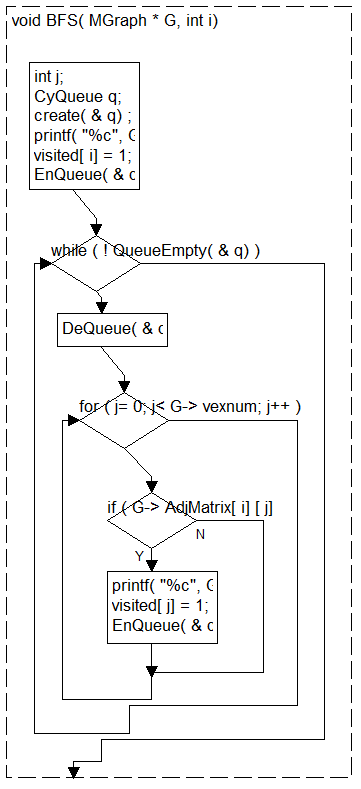
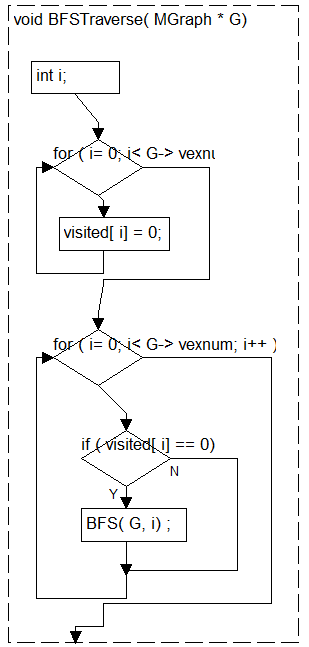
输入数据：采用合适的方式创建上图，如先输入顶点数 n、

边数 e，然后依次输入边信息；遍历时要求从键盘输入起始顶点。

输出形式：分别输出给定起始顶点的深度优先遍历序列和广度优先遍历序列。

3. 算法描述

给出算法描述的程序流程图



4. 程序设计

给出实现程序功能的函数及说明，示例如下：

(1) 本程序包含的函数

AMGraph CreateUDN(); // 创建用于DFS的图

void BFSTraverse(MGraph \*G); // 初始化

void BFS(MGraph \*G, int i); // 搜索

int QueueEmpty(CyQueue \*q); // 检查队列是否已满

void DeQueue(CyQueue \*q, dataType \*value); // 出队

void EnQueue(CyQueue \*q, dataType value); // 入队

void create(CyQueue \*q); // 创建队列

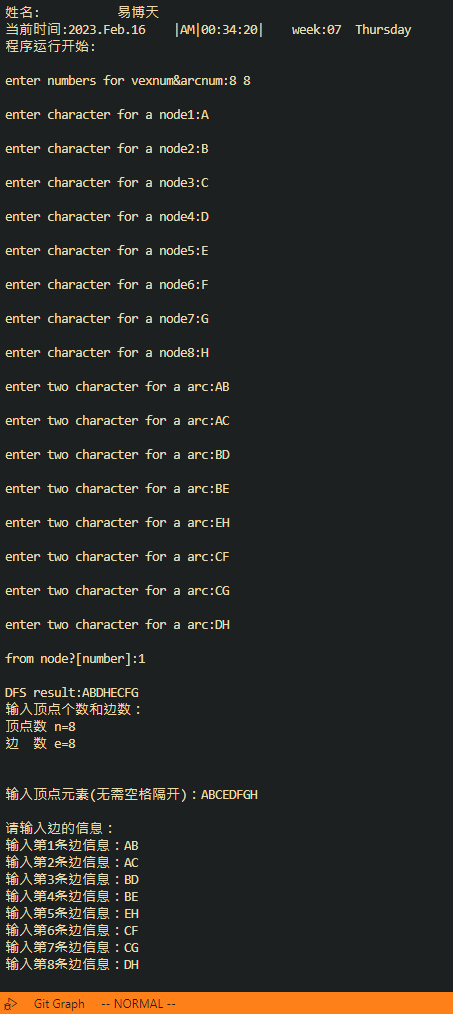
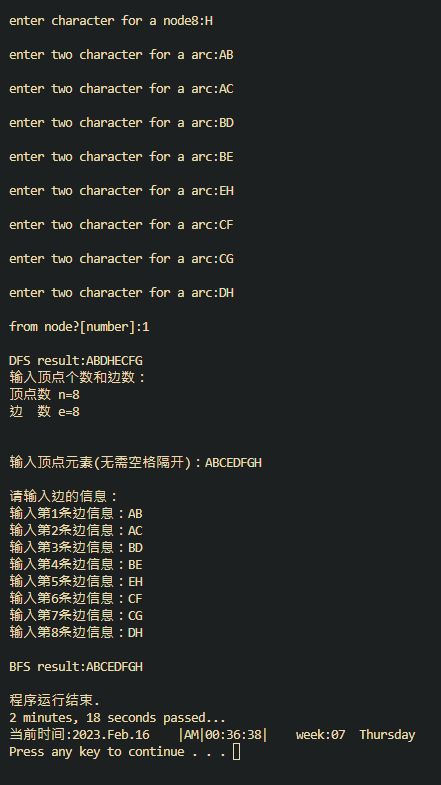
void CreateUDG(MGraph \*G); // 创建用于BFS的图

int LocateVex2(MGraph \*G, VertexType v); // 查找元素v在一维数组 Vertex[] 中的下标，并返回下标

int main()

### 四、调试分析

#### 1、调试过程及说明

* 以截屏图像的方式给出，包括程序运行开始和结束时的系统时间、姓名、学号、实验名称等信息.
* 

#### 2、调试分析

如图

### 五、使用说明

按照运行时提示进行输入

**六、测试结果**

内容：给出测试用例测试程序。写出测试的情况，使用的输入以及得到的结果。

要求：测试的情况越全面越好。

如上图所示

**七、实验的收获和感想**

其实还可以复用，其中相当一部分代码可以大大简化

**八、附录：源代码（双栏排版，加必要的注释）**

#include "homework.h"

#define true 1

#define MVNum 100

#define VertexMax 100 // 最大顶点数为100

#define Maxsize 100 // 队列最大元素个数100

typedef char VertexType; // 每个顶点数据类型为字符型

typedef int dataType; // 队列元素类型

/\*图结构体\*/

typedef struct

{

VertexType Vertex[VertexMax]; // 存放顶点元素的一维数组

int AdjMatrix[VertexMax][VertexMax]; // 邻接矩阵二维数组

int vexnum, arcnum; // 图的顶点数和边数

} MGraph;

/\*队列结构体\*/

typedef struct

{

dataType \*base;

int front;

int rear;

} CyQueue;

typedef struct

{

char vexs[MVNum];

int arcs[MVNum][MVNum];

int vexnum, arcnum;

} AMGraph;

AMGraph CreateUDN(); // 创建用于DFS的图

void BFSTraverse(MGraph \*G); // 初始化

void BFS(MGraph \*G, int i); // 搜索

int QueueEmpty(CyQueue \*q); // 检查队列是否已满

void DeQueue(CyQueue \*q, dataType \*value); // 出队

void EnQueue(CyQueue \*q, dataType value); // 入队

void create(CyQueue \*q); // 创建队列

void CreateUDG(MGraph \*G); // 创建用于BFS的图

int LocateVex2(MGraph \*G, VertexType v); // 查找元素v在一维数组 Vertex[] 中的下标，并返回下标

int main()

{

homework\_start();

AMGraph G1;

G1 = CreateUDN();

int where;

printf("\nfrom node?[number]:");

scanf("%d", &where);

fflush(stdin);

printf("\nDFS result:");

DFS\_AM(G1, where);

MGraph G2;

CreateUDG(&G2);

printf("\nBFS result:");

BFSTraverse(&G2);

homework\_done();

return 0;

}

/\*无向图UDG的基本操作\*/

int LocateVex2(MGraph \*G, VertexType v) // 查找元素v在一维数组 Vertex[] 中的下标，并返回下标

{

int i;

for (i = 0; i < G->vexnum; i++)

{

if (v == G->Vertex[i])

{

return i;

}

}

printf("No Such Vertex!\n");

return -1;

}

void CreateUDG(MGraph \*G)

{

int i, j;

printf("\n输入顶点个数和边数：\n");

printf("顶点数 n=");

scanf("%d", &G->vexnum);

printf("边 数 e=");

scanf("%d", &G->arcnum);

printf("\n");

printf("\n");

printf("输入顶点元素(无需空格隔开)：");

scanf("%s", G->Vertex);

printf("\n");

for (i = 0; i < G->vexnum; i++)

for (j = 0; j < G->vexnum; j++)

{

G->AdjMatrix[i][j] = 0;

}

int n, m;

VertexType v1, v2;

printf("请输入边的信息：\n");

for (i = 0; i < G->arcnum; i++)

{

printf("输入第%d条边信息：", i + 1);

scanf(" %c%c", &v1, &v2);

n = LocateVex2(G, v1);

m = LocateVex2(G, v2);

if (n == -1 || m == -1)

{

printf("NO This Vertex!\n");

return;

}

G->AdjMatrix[n][m] = 1;

G->AdjMatrix[m][n] = 1;

}

}

/\*循环队列基本操作\*/

void create(CyQueue \*q)

{

q->base = (dataType \*)malloc(Maxsize \* sizeof(dataType));

if (!q->base)

{

printf("Space allocation failed!\n");

return;

}

q->front = q->rear = 0;

return;

}

void EnQueue(CyQueue \*q, dataType value)

{

if ((q->rear + 1) % Maxsize == q->front)

{

printf("Cyclic Queue is Full!\n");

return;

}

q->base[q->rear] = value;

q->rear = (q->rear + 1) % Maxsize;

return;

}

void DeQueue(CyQueue \*q, dataType \*value)

{

if (q->front == q->rear)

{

printf("Cyclic Queue is Empty!\n");

return;

}

\*value = q->base[q->front];

q->front = (q->front + 1) % Maxsize;

return;

}

int QueueEmpty(CyQueue \*q)

{

if (q->front == q->rear) // 队列为空返回1，不为空返回0

{

return 1;

}

return 0;

}

/\*广度优先遍历BFS\*/

int visited[VertexMax]; // 定义"标志"数组为全局变量

void BFS(MGraph \*G, int i)

{

int j;

CyQueue q;

create(&q);

// 1.设置起始点

printf("%c", G->Vertex[i]); // 1.输出当前结点

visited[i] = 1; // 2.将已访问的结点标志成1

EnQueue(&q, i); // 3.将第一个结点入队

// 2.由起始点开始，对后续结点进行操作

while (!QueueEmpty(&q))

{

DeQueue(&q, &i);

for (j = 0; j < G->vexnum; j++)

{

if (G->AdjMatrix[i][j] == 1 && visited[j] == 0)

{

printf("%c", G->Vertex[j]); // 输出符合条件的顶点

visited[j] = 1; // 设置成已访问状态1

EnQueue(&q, j); // 入队

}

}

}

}

void BFSTraverse(MGraph \*G)

{

int i;

// 数组初始化为全0

for (i = 0; i < G->vexnum; i++)

{

visited[i] = 0;

}

for (i = 0; i < G->vexnum; i++)

{

if (visited[i] == 0)

{

BFS(G, i);

}

}

}

int LocateVex(AMGraph G, char v)

{

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

{

if (G.vexs[i] == v)

return i;

}

}

AMGraph CreateUDN()

{

AMGraph G;

printf("enter numbers for vexnum&arcnum:");

scanf("%d%d", &G.vexnum, &G.arcnum); // vn是点数，an是边数

fflush(stdin);

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

{

static int node = 1;

printf("\nenter character for a node%d:", node++);

scanf("%c", &G.vexs[i]);

fflush(stdin);

}

char v1, v2;

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

{

printf("\nenter two character for a arc:");

scanf("%c%c", &v1, &v2);

fflush(stdin);

int i = LocateVex(G, v1);

int j = LocateVex(G, v2);

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

G.arcs[j][i] = G.arcs[i][j]; // 无向对称阵

}

return G;

}

int visited\_DFS[MVNum] = {0};

void DFS\_AM(AMGraph G, int v)

{

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

visited\_DFS[v] = true;

for (int col = 1; col <= G.vexnum; col++)

{

if ((G.arcs[v][col] == 1) && (!visited\_DFS[col]))

DFS\_AM(G, col);

}

}