```c
// 2020240013 大数据201 向豪

#include <malloc.h>
#include <stdio.h>

#define MAXSIZE 1000
#define MAX_AMNUM 100

// 枚举 布尔
typedef enum {
    FALSE,
    TRUE
} Boolean;

// 队列
typedef struct {
    int *base;
    int front;
    int rear;
} SqQueue;

// 无向图
typedef struct {
    char verxs[MAX_AMNUM];
    int arcs[MAX_AMNUM][MAX_AMNUM];
    int numVertexes, numEdges;
} AMGraph;

Boolean InitAMGraph(AMGraph *G); // 初始化
void CreateAMGraph(AMGraph *G); // 创建无向图
void PrintAMatrix(AMGraph G); // 打印无向图
void DFS_AM(AMGraph G, int v); // 连通 深度遍历
void DFSTraverse(AMGraph G, int v); // 非连通 深度遍历
void BFS_AM(AMGraph G, int v); // 连通 广度遍历
void BFSTraverse(AMGraph G, int v); // 非连通 广度遍历
Boolean InitQueue(SqQueue *queue); // 队列初始化
Boolean EnQueue(SqQueue *queue, int elem); // 入队
Boolean DeQueue(SqQueue *queue, int *elem); // 出队
Boolean IsFull(SqQueue *queue); // 判队满
Boolean IsEmpty(SqQueue *queue); // 判队空

// 访问标志
Boolean visited[MAX_AMNUM];

int main() {
```

```
    // 定义图
    AMGraph G;
    // 初始化
    InitAMGraph(&G);
    // 创建无向图
    CreateAMGraph(&G);
    // 打印无向图
    printf("======= 无向图 G 的邻接矩阵 =======\n");
    PrintAMatrix(G);
    printf("================================\n");
    // 深度遍历
    printf("无向图 G 以 V1 为源点的 DFS 遍历序列为: \n");
    DFSTraverse(G, 1);
    printf("\n");
    printf("无向图 G 以 V3 为源点的 BFS 遍历序列为: \n");
    BFSTraverse(G, 3);

    return 0;
}

Boolean InitAMGraph(AMGraph *G) {
    G = (AMGraph *) malloc(sizeof(AMGraph));
    if (!G) {
        return FALSE;
    }
    return TRUE;
}

void CreateAMGraph(AMGraph *G) {
    // 顶点数+边数
    printf("输入无向图的顶点数和边数，用空格分开: ");
    scanf("%d %d", &(G->numVertexes), &(G->numEdges));
    // 顶点编号
    for (int i = 0; i < G->numVertexes; i++) {
        G->verxs[i] = i;
    }
    // 初始化 无边为 0
    for (int i = 0; i < G->numVertexes; i++) {
        for (int j = 0; j < G->numVertexes; j++) {
            G->arcs[i][j] = 0;
        }
    }
    PrintAMatrix(*G);
    // 输入边信息
```

```
    int sub1, sub2;
    printf("输入%d 条边（邻接矩阵下三角下标对，空格分开）：\n", G->numEdges);
    for (int i = 0; i < G->numEdges; i++) {
        printf("第%d 条边：", (i + 1));
        scanf("%d %d", &sub1, &sub2);
        G->arcs[sub1][sub2] = G->arcs[sub2][sub1] = 1; //有边为 1
    }
}

void PrintAMatrix(AMGraph G) {
    // 表头
    printf("    _");
    for (int i = 0; i < G.numVertexes; i++) {
        printf("V%d ", i);
    }
    printf("_");
    printf("\n");
    // 内容
    for (int i = 0; i < G.numVertexes; i++) {
        if (i == (G.numVertexes - 1)) {
            printf("V%d |_", i); // 尾行
        } else {
            printf("V%d | ", i);
        }
        for (int j = 0; j < G.numVertexes; j++) {
            printf(" %d ", G.arcs[i][j]);
        }
        if (i == (G.numVertexes - 1)) {
            printf("_|"); // 尾行
        } else {
            printf(" |");
        }
        printf("\n");
    }
}

void DFS_AM(AMGraph G, int v) {
    // 打印遍历顶点
    printf("V%d ", v);
    visited[v] = TRUE; // 遍历过
    for (int i = 0; i < G.numVertexes; i++) {
        // 行为相邻，1 为有相邻，且未走过
        if ((G.arcs[v][i] != 0) && (!visited[i])) {
            DFS_AM(G, i); // 相邻点放入，递归遍历
        }
```

```
    }
}

void DFSTraverse(AMGraph G, int v) {
    // 初始化 visited[]标志数组，防止影响其他遍历
    for (int i = 0; i < G.numVertexes; i++) {
        visited[i] = FALSE;
    }
    for (int i = 0; i < G.numVertexes; i++) {
        // 如果还有未遍历完顶点，一定是非连通。则跳转到非连通顶点重复 DFS_AM()遍历
        if (!visited[i]) {
            DFS_AM(G, v); // 从 顶点 v 开始遍历
        }
    }
}

void BFS_AM(AMGraph G, int v) {
    printf("V%d ", v); // 打印遍历顶点
    visited[v] = TRUE; // 遍历过
    // 这里需要借用 队列操作
    SqQueue queue; // 定义
    InitQueue(&queue); // 初始化
    EnQueue(&queue, v); // v 入队列

    int u;
    // 循环，队列出完为止
    while (!IsEmpty(&queue)) {
        DeQueue(&queue, &u); // 队头元素出队列，用 u 来装，取出来供以下循环使用
        for (int i = 0; i < G.numVertexes; i++) {
            // 行为相邻，1 为有相邻，且未走过
            if ((G.arcs[u][i] != 0) && (!visited[i])) { // for 遍历完 u 行为止
                printf("V%d ", i); // 打印遍历顶点
                visited[i] = TRUE; // 遍历过
                EnQueue(&queue, i); // i 入队列
            }
        }
    }
}

void BFSTraverse(AMGraph G, int v) {
    // 初始化 visited[]标志数组，防止影响其他遍历
    for (int i = 0; i < G.numVertexes; i++) {
        visited[i] = FALSE;
    }
    for (int i = 0; i < G.numVertexes; i++) {
```

```
        // 如果还有未遍历完顶点，一定是非连通。则跳转到非连通顶点重复 BFS_AM()遍历
        if (!visited[i]) {
            BFS_AM(G, v); // 从 顶点 v 开始遍历
        }
    }
}

Boolean InitQueue(SqQueue *queue) {
    // 数据指针 指向申请的 1000 个数据空间
    queue->base = (int *) malloc(sizeof(int) * MAXSIZE);
    if (!(queue->base)) {
        return FALSE;
    }
    // 头指向数组 0 位置。尾指针指向头指针位置，队列为空
    queue->front = queue->rear = 0;
    return TRUE;
}

Boolean EnQueue(SqQueue *queue, int elem) {
    // 判队满
    if (IsFull(queue)) {
        return FALSE;
    }
    // 尾指针
    queue->base[queue->rear] = elem; // 赋值
    queue->rear = (queue->rear + 1) % MAXSIZE; // 向后移动
    return TRUE;
}

Boolean DeQueue(SqQueue *queue, int *elem) {
    // 判队空
    if (IsEmpty(queue)) {
        return FALSE;
    }
    *elem = queue->base[queue->front]; // 取出队头
    queue->front = (queue->front + 1) % MAXSIZE; // 队头变为原队头的下一个位置
    return TRUE;
}

Boolean IsFull(SqQueue *queue) {
    // 尾指针向后移动一个位置后与头指针重叠，此时队满
    return (queue->rear + 1) % MAXSIZE == queue->front;
}

Boolean IsEmpty(SqQueue *queue) {
```

```
    // 尾指针和头指针重叠时，此时队空
    return queue->rear == queue->front;
}
```