**测试用例：** //第一行 8表示节点数，10表示边数，第一行后，为两个数字，最后一个为0的，为孤立结点

8 10

1 2 3

2 1 4 5

3 1 6 7

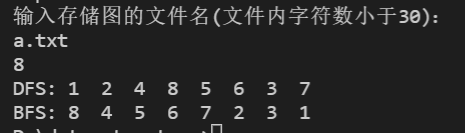
4 2 8

5 2 8

6 3 8

7 3 8

8 4 5 6 7



**算法和数据结构设计：**

i）创建图。单独读取第一行的结点数和边数。之后，每一行的第一个数字作为读取到的结点，利用链表，将每一行后面的数字链接起来，表示和这个结点相连的点，每个链表最后位置置空

ii）深度遍历。利用递归，用标识visited表示是否访问过这个结点，0表示未访问过，1表示访问过。先将访问到的这个结点输出，并标记visited = 1。从这个结点依次向下，若结点已经访问过，继续下一个结点，若没有，进入下一层递归。直到这一深度的所有结点都已经访问过。

iii）广度遍历。利用递归和队列（队列记录某一层次访问过的结点）。先将最开始的结点输出，进入遍历过程。将这个结点标记访问过，接着访问和这一结点邻接的所有结点，输出，并加入队列中。如果已经访问过了所有结点，结束递归。若没有，从队列中弹出一个结点，从这个结点继续递归过程。

#include <stdio.h>

#include <stdlib.h>

#define FILE\_NAME\_LEN 30

#define NODE\_NUM 20

#define LEN sizeof(struct graphnode)

typedef struct graphnode {

    int data;

    struct graphnode \*link;

}Node;

typedef struct que {

    int data[NODE\_NUM];

    int count;

    int front;

    int rear;

}Queue;

void CreatG(Node graph[], FILE \*fp, int num[]);

void DFS(Node graph[], int viseted[], int v);

void BFS(Node graph[], int visited[], Queue \*que, int i, int v);

void In(int d, Queue \*que, int v);

int Out(Queue \*que, int v);

int main() {

    // open file

    char file\_name[FILE\_NAME\_LEN];

    printf("输入存储图的文件名(文件内字符数小于%d)：\n", FILE\_NAME\_LEN);

    scanf("%s", file\_name);

    FILE \*fp = fopen(file\_name, "r");

    if (fp == NULL) {

        printf("打开文件失败。");

        exit(-1);

    }

    Node graph[NODE\_NUM];

    int num[2], i, v0 = 0; //num[0] 表示总节点数，num[1] 表示总边数

    CreatG(graph, fp, num);   //文件每行末尾没有空格

    //DFS

    int visited[num[0]]; // 0:未访问过, 1:访问过

    for(i = 0; i < num[0]; i++)

        visited[i] = 0;

    printf("\nDFS: ");

    DFS(graph, visited, v0);

    //BFS

    for (i = 0; i < num[0]; i++)

        visited[i] = 0;

    Queue \*que;

    que = (Queue\*)malloc(sizeof(Queue));

    que->front = 0;

    que->rear = 0;

    que->count = 1;

    int v = num[0];

    printf("\nBFS: ");

    printf("%d  ", graph[num[0] - 1].data);

    i = num[0] - 1;

    BFS(graph, visited, que, i, v);

    fclose(fp);

    return 0;

}

void BFS(Node graph[], int visited[], Queue \*que, int i, int v) {

    visited[i] = 1;

    Node \*p;

    p = (Node\*)malloc(sizeof(Node));

    p = graph[i].link;

    while(p != NULL) {

        if(visited[p->data - 1] == 0) {

            printf("%d  ", p->data);

            visited[p->data - 1] = 1;

            que->count++;

            In(p->data, que, v);

        }

        p = p->link;

    }

    if(que->count < v) {

        i = Out(que, v);

        BFS(graph, visited, que, i, v);

    }

}

void In(int d, Queue \*que, int v) {

    que->rear++;

    if(que->rear == v)

        que->rear = 0;

    if(que->front != que->rear)

        que->data[que->rear] = d;

    else printf("the queue is full");

}

int Out(Queue \*que, int v) {

    int data;

    if(que->rear == que->front){

        printf("the queue is empty");

        exit;

    }

    que->front++;

    if(que->front == v)

        que->front = 0;

    data = que->data[que->front] - 1;

    return data;

}

void DFS(Node graph[], int visited[], int v) {

    Node \*p;

    printf("%d  ", graph[v].data);

    visited[v] = 1;

    p = graph[v].link;

    while(p != NULL) {

        if(visited[p->data - 1] == 0) {

            DFS(graph, visited, p->data - 1);

        }

        p = p->link;

    }

}

void CreatG(Node graph[], FILE \*fp, int a[]) {

    int i = 0, c;  //a[0]表示结点总数，a[1]表示边数,i表示当前纪录到的结点数,c暂时记录信息

    Node \*p1, \*p2, \*head; //搜索指针

    fscanf(fp, "%d%d", &a[0], &a[1]);

    fgetc(fp);  //读取  '\n'

    while(1) {

        head = (Node \*)malloc(LEN);

        head->link = NULL;

        p1 = head;

        fscanf(fp, "%d", &graph[i].data);

        fgetc(fp);

        while(1) {

            p2 = (Node\*)malloc(LEN);

            fscanf(fp, "%d", &p2->data);

            p1->link = p2;

            p1 = p2;

            if((c = fgetc(fp)) == '\n' || c == -1) {

                p1->link = NULL;

                break;

            }

        }

        graph[i].link = head->link;

        free(head);

        i++;

        if(c == -1) {

            break;

        }

    }

    printf("%d", graph[7].data);

}