

## 《数据结构与算法》实验报告

学生姓名		院（系）	计算机科学与技术
学 号		专 业	
实验时间		实验地点	
实验项目	实验 3/4：图型结构的建立与搜索		
<p><b>实验目的：</b>将课程的基本原理、技术和方法与实际应用相结合，训练和提高学生组织、存储和处理信息的能力，以及复杂问题的数据结构设计能力和程序设计能力，培养软件设计与开发所需要的实践能力。</p> <p><b>实验要求：</b>灵活运用基本的数据结构和算法知识，对实际问题进行分析和抽象；结合程序设计的一般过程和方法为实际问题设计数据结构和有效算法；用高级语言对数据结构和算法进行编程实现、调试，测试其正确性和有效性。</p>			
<p><b>实验内容：</b></p> <p>图的搜索（遍历）算法是图型结构相关算法的基础，本实验要求编写程序演示图两种典型存储结构的建立和搜索（遍历）过程。</p> <ol style="list-style-type: none"> <li>1) 分别实现图的邻接矩阵、邻接表存储结构的建立算法，分析和比较各建立算法的时间复杂度以及存储结构的空间占用情况；</li> <li>2) 实现图的邻接矩阵、邻接表两种存储结构的相互转换算法；</li> <li>3) 在上述两种存储结构上，分别实现图的深度优先搜索（递归和非递归）和广度优先搜索算法。并以适当的方式存储和显示相应的搜索结果（深度优先或广度优先生成森林（或生成树）、深度优先或广度优先序列和编号）；</li> <li>4) 分析搜索算法的时间复杂度；</li> <li>5) 以文件形式输入图的顶点和边，并显示相应的结果。要求顶点不少于 10 个，边不少于 13 个；</li> <li>6) 软件功能结构安排合理，界面友好，便于使用。</li> </ol>			
<p><b>数据结构定义：</b></p> <pre>int visited[MAX_NUM]; /*访问记录*/  int dfn[MAX_NUM]; /*深度优先编号*/  int bfn[MAX_NUM]; /*广度优先编号*/  int cnt; /*遍历编号*/</pre>			

```

/*定义队列*/
typedef struct Queue
{
    int queue[MAX_NUM];
    int front;
    int rear;
}Queue;

/*定义图种类*/
typedef enum{DG,AG} GraphKind;

/*定义邻接矩阵的每个元素属性*/
typedef struct ArcCell
{
    int adj;    /*标记两点是否有边*/
    int info;   /*权值*/
}ArcCell , AdjMatrix[MAX_NUM][MAX_NUM];    /*定义为矩阵*/

/*定义无向图的邻接矩阵整体*/
typedef struct Mgraph
{
    char element[MAX_NUM]; /*保存顶点数据*/
    AdjMatrix arcs;        /*邻接关系*/
    int vexnum, arcnum;    /*顶点数量，边数量*/
    GraphKind kind;        /*图的种类*/
}MGraph;

/*定义邻接表的结点属性*/
typedef struct ArcNode
{
    int adjvex; /*标记该点的下标*/
    struct ArcNode * next; /*链接*/
    int info; /*权值*/
}

```

```

}ArcNode;

/*定义头结点列表*/
typedef struct Vnode /*定义头节点*/
{
    char element; /*头结点顶点数据域*/
    ArcNode * firstnode; /*头指针*/
}Vnode, AdjList[MAX_NUM]; /*定义为列表*/

/*定义无向图邻接表整体*/
typedef struct ALGraph
{
    AdjList vertices; /*定义头结点列表*/
    int vexnum,arcnum; /*定义顶点数量，边数量*/
    GraphKind kind; /*定义图种类*/
}ALGraph;

```

算法设计与分析（要求画出核心内容的程序流程图）：

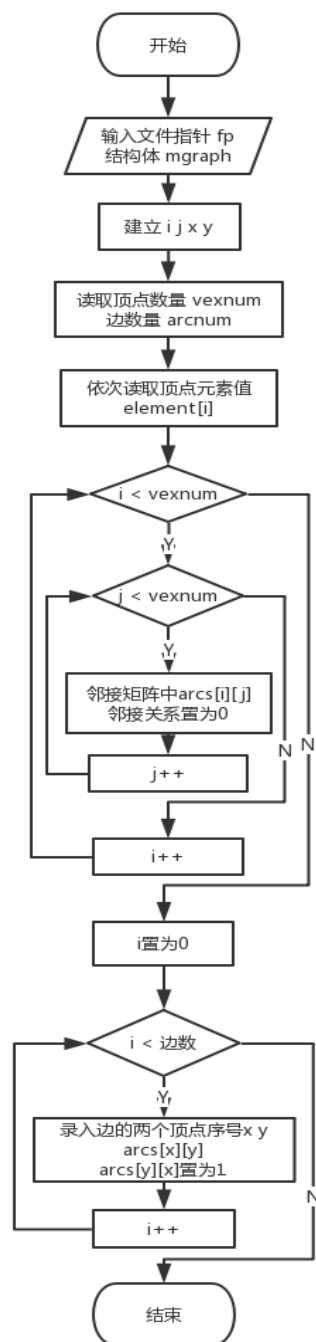
### 1. 邻接矩阵和邻接表建立算法的时间复杂度和空间占用情况

邻接矩阵建立的时间复杂度为  $O(n^2)$ ，由于需要建立一个  $n * n$  矩阵，需要占用的空间为  $|n|^2$  ( $n$  为顶点个数)。

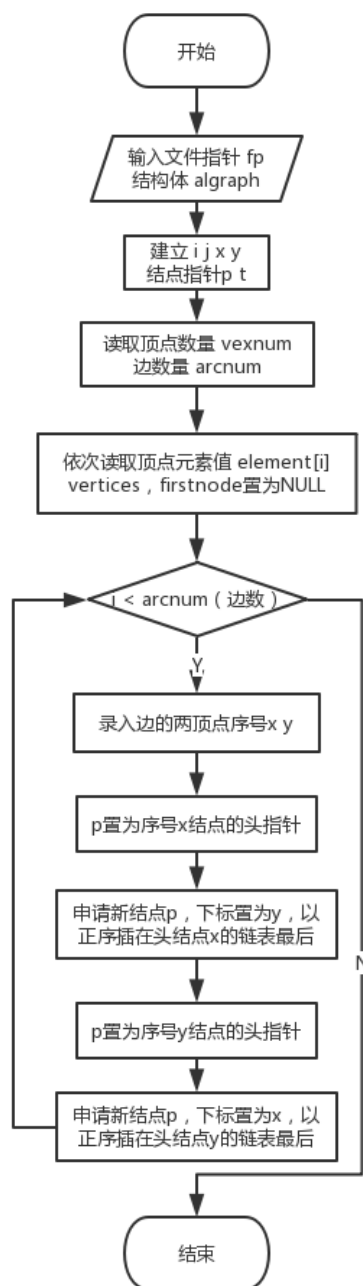
邻接表建立的时间复杂度为  $O(n + e)$ ，空间占用为  $|n| + |e|$ 。（ $e$  为边数）

邻接矩阵建立的时间复杂度和空间复杂度均大于邻接表，如果需要频繁访问一些边，则采用邻接矩阵，此时时间开销固定。

邻接表存储空间小于邻接矩阵，不存储无边情况，如果  $E \ll V^2$ ，则可以采用邻接表。下面是两种建立算法。



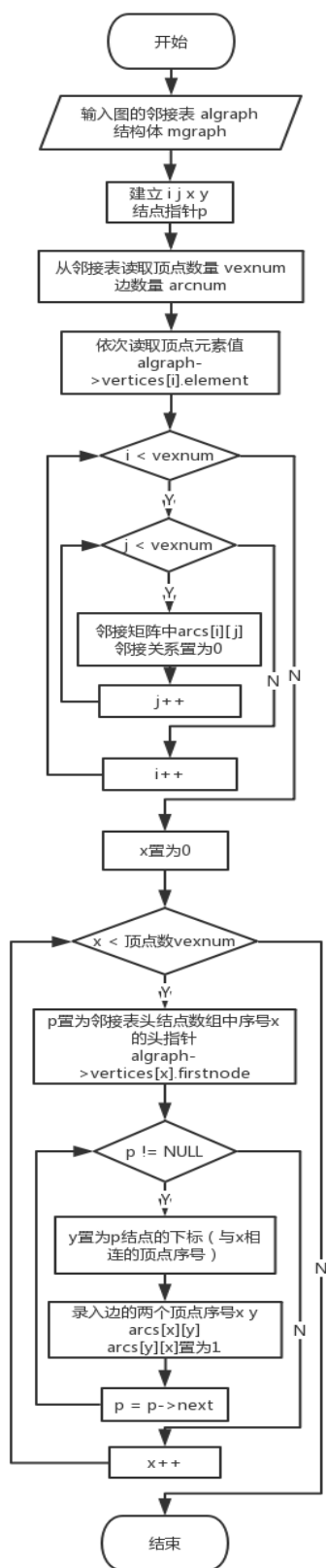
邻接矩阵建立



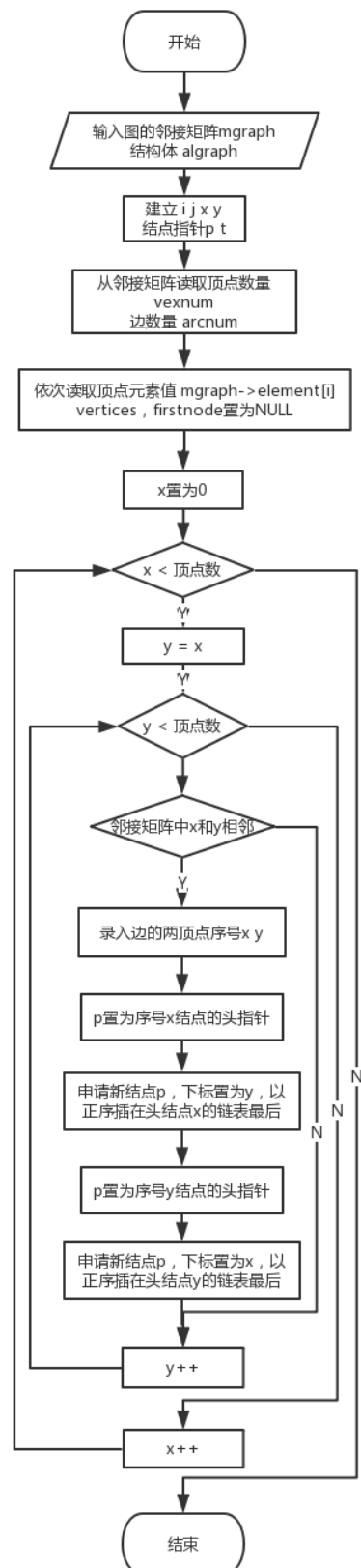
邻接表建立

## 2. 两种结构相互转换

相互转换算法就是将两种结构的建立算法的数据输入源从文件改为另一个图的信息源输入。过程大致与图的建立算法相同。以下是两种转换算法。



邻接表转邻接矩阵

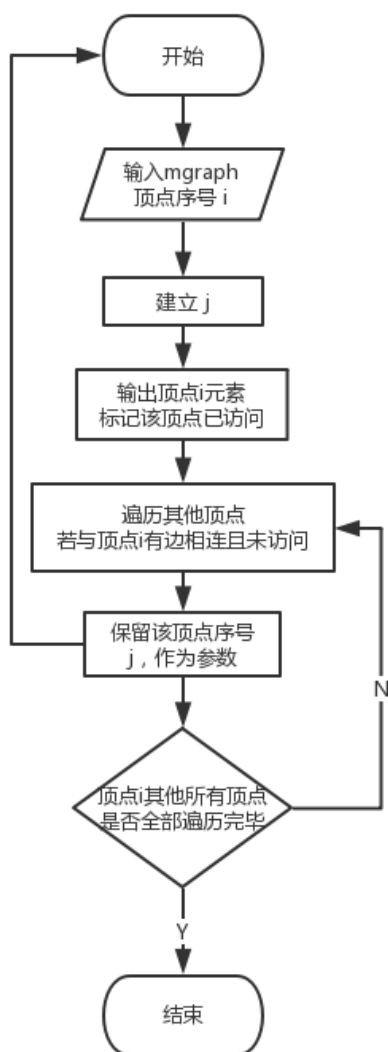


邻接矩阵转邻接表

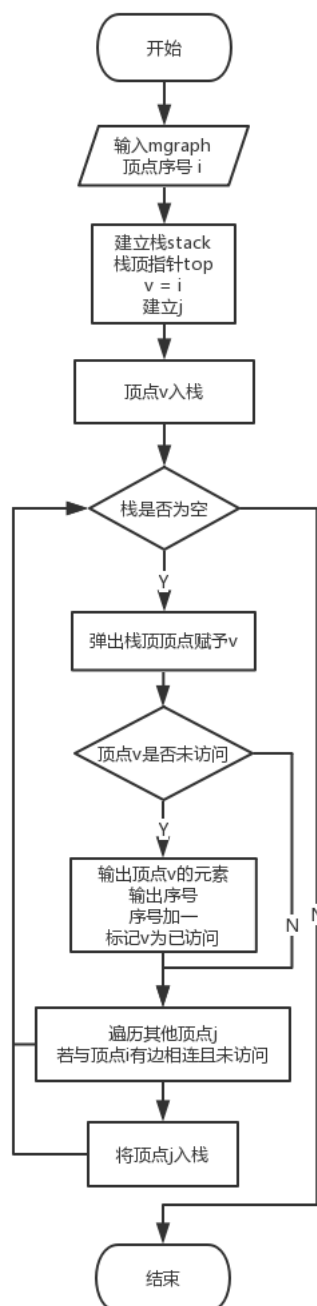
### 3. 邻接矩阵的 DFS、BFS 实现，显示和时间复杂度

邻接矩阵的深度优先搜索和广度优先搜索的时间复杂度均为  $O(n^2)$ , 深度优先搜索时, 每入栈一个顶点, 需要对其他所有顶点遍历搜索是否关联,  $n$  个顶点都要入栈, 因此 DFS 时间复杂度为  $O(n^2)$ 。BFS 将栈换为队列, 同理, 时间复杂度为  $O(n^2)$ 。

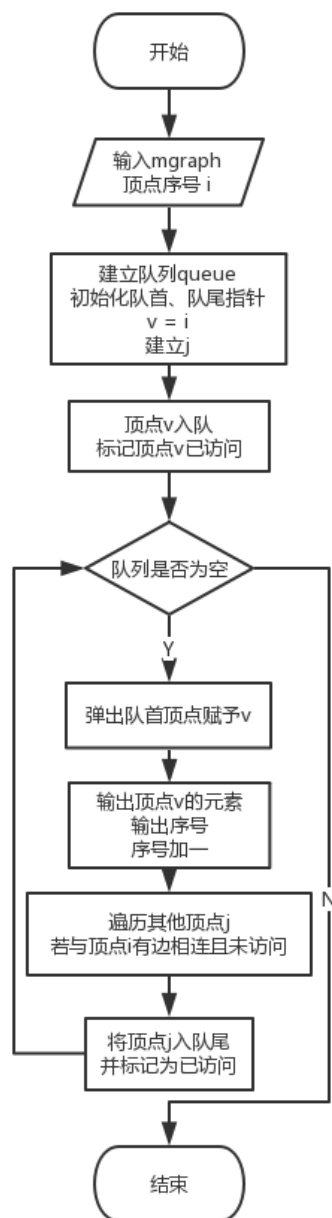
为了以适当的形式显示 DFS 和 BFS 序列, 设立 dfn 和 bfn 数组, 每当在栈中或队列中弹出顶点时, 在顶点对应序号的下标保存遍历序号并且在输出元素时将 DFS 或 BFS 序号一同输出。以下是算法流程图



DFS 递归算法



DFS 非递归算法

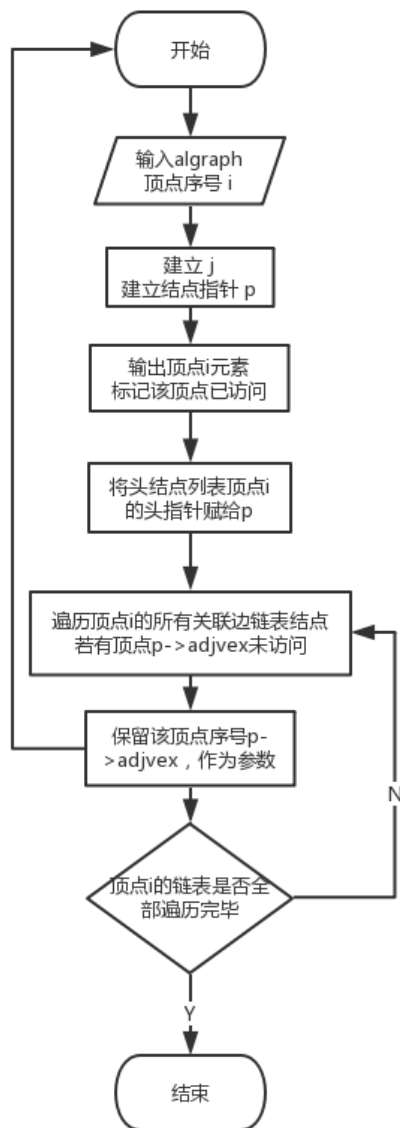


BFS 算法

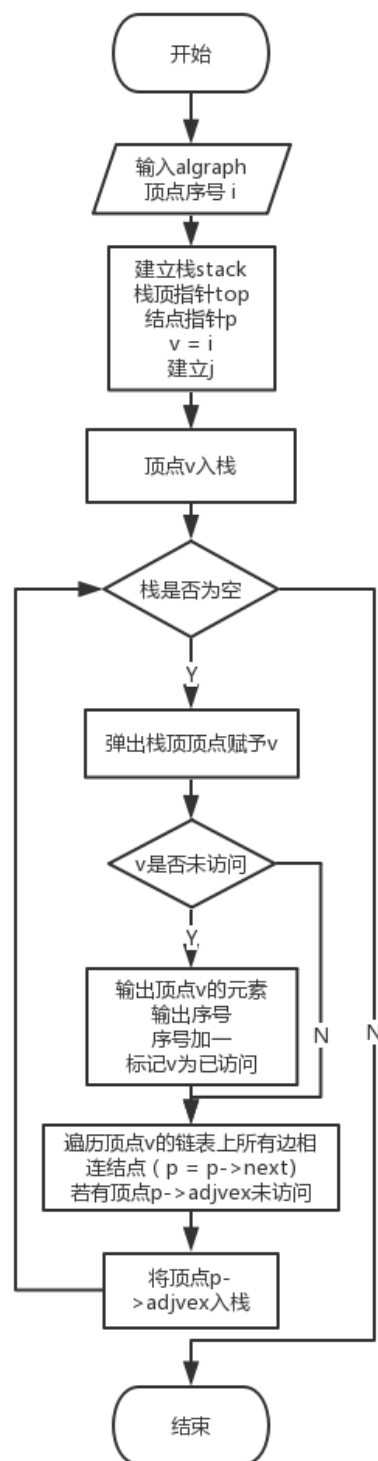
#### 4. 邻接表的 DFS、BFS 实现，显示和时间复杂度

邻接表的深度优先搜索和广度优先搜索的时间复杂度均为  $O(n+e)$ , 因为邻接表中只保存与顶点相连的其他顶点序号。所以 DFS 时，入栈的每个顶点遍历它的相邻边  $e_1$ ,  $n$  个顶点都要入栈，总的时间复杂度为  $n + e_1 + \dots + e_n = O(n+e)$ 。BFS 将栈换为队列，同理，复杂度为  $O(n+e)$ 。

为了以适当的形式显示 DFS 和 BFS 序列，设立  $dfn$  和  $bfn$  数组，每当在栈中或队列中弹出顶点时，在顶点对应序号的下标保存遍历序号并且在输出元素时将 DFS 或 BFS 序号一同输出。以下是算法流程图

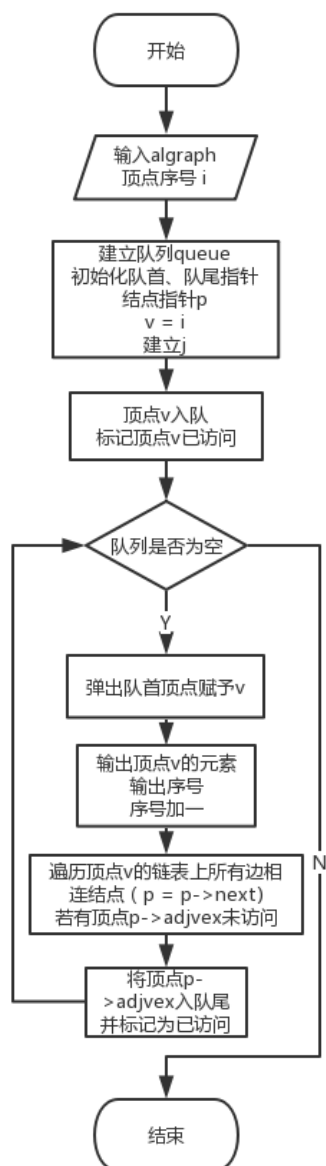


邻接表的 DFS 递归



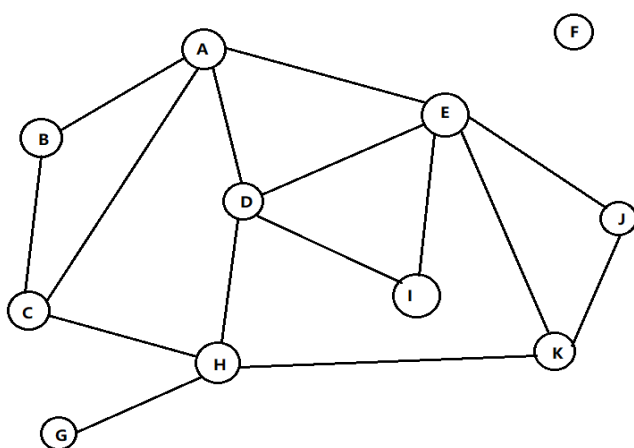
邻接表的 DFS 非递归





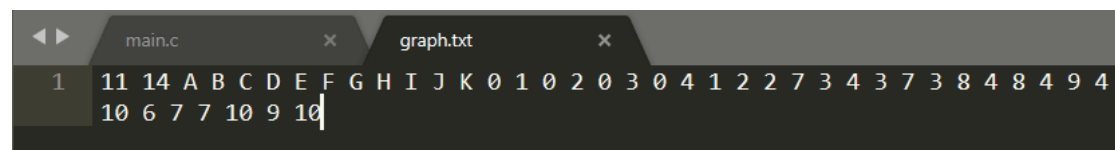
邻接表的 BFS

实验测试结果及结果分析：  
测试用例



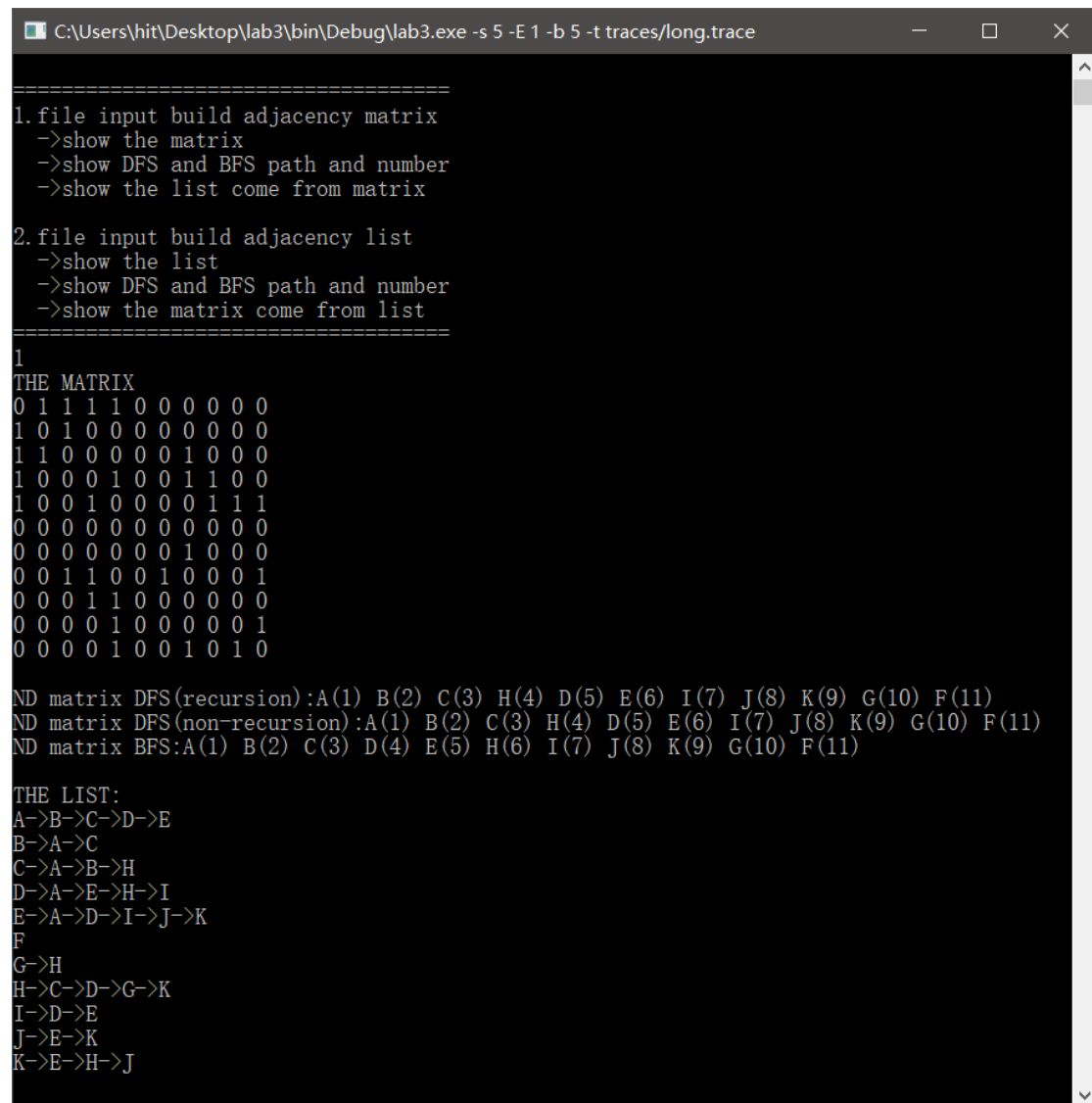
11 个顶点，15 条边

## 文件输入



```
1 11 14 A B C D E F G H I J K 0 1 0 2 0 3 0 4 1 2 2 7 3 4 3 7 3 8 4 8 4 9 4
10 6 7 7 10 9 10
```

## 邻接矩阵形式



```
C:\Users\hit\Desktop\lab3\bin\Debug\lab3.exe -s 5 -E 1 -b 5 -t traces/long.trace

=====
1. file input build adjacency matrix
   ->show the matrix
   ->show DFS and BFS path and number
   ->show the list come from matrix

2. file input build adjacency list
   ->show the list
   ->show DFS and BFS path and number
   ->show the matrix come from list
=====
1
THE MATRIX
0 1 1 1 1 0 0 0 0 0 0
1 0 1 0 0 0 0 0 0 0 0
1 1 0 0 0 0 0 1 0 0 0
1 0 0 0 1 0 0 1 1 0 0
1 0 0 1 0 0 0 0 1 1 1
0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0
0 0 1 1 0 0 1 0 0 0 1
0 0 0 1 1 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 1
0 0 0 0 1 0 0 1 0 1 0

ND matrix DFS(recursion):A(1) B(2) C(3) H(4) D(5) E(6) I(7) J(8) K(9) G(10) F(11)
ND matrix DFS(non-recursion):A(1) B(2) C(3) H(4) D(5) E(6) I(7) J(8) K(9) G(10) F(11)
ND matrix BFS:A(1) B(2) C(3) D(4) E(5) H(6) I(7) J(8) K(9) G(10) F(11)

THE LIST:
A->B->C->D->E
B->A->C
C->A->B->H
D->A->E->H->I
E->A->D->I->J->K
F
G->H
H->C->D->G->K
I->D->E
J->E->K
K->E->H->J
```

显示矩阵、DFS、BFS 以及相关遍历序列编号和转换的邻接表

## 邻接表形式

```
C:\Users\hit\Desktop\lab3\bin\Debug\lab3.exe -s 5 -E 1 -b 5 -t traces/long.trace

=====
1.file input build adjacency matrix
->show the matrix
->show DFS and BFS path and number
->show the list come from matrix
=====
2.file input build adjacency list
->show the list
->show DFS and BFS path and number
->show the matrix come from list
=====
2
THE LIST:
A->B->C->D->E
B->A->C
C->A->B->H
D->A->E->H->I
E->A->D->I->J->K
F
G->H
H->C->D->G->K
I->D->E
J->E->K
K->E->H->J

ND list DFS(recursion):A(1) B(2) C(3) H(4) D(5) E(6) I(7) J(8) K(9) G(10) F(11)
ND list DFS(non-recursion):A(1) E(2) K(3) J(4) H(5) G(6) D(7) I(8) C(9) B(10) F(11)
ND list BFS:A(1) B(2) C(3) D(4) E(5) H(6) I(7) J(8) K(9) G(10) F(11)

THE MATRIX:
0 1 1 1 1 0 0 0 0 0 0
1 0 1 0 0 0 0 0 0 0 0
1 1 0 0 0 0 0 1 0 0 0
1 0 0 0 1 0 0 1 1 0 0
1 0 0 1 0 0 0 0 1 1 1
0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0
0 0 1 1 0 0 1 0 0 0 1
0 0 0 1 1 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 1
0 0 0 0 1 0 0 1 0 1 0
```

显示邻接表、DFS、BFS 以及相关遍历序列编号和转换的矩阵

问题及解决方法:

1. 如何设定文件输入的格式?  
顶点数-边数-各顶点元素名称-各边两端顶点编号
2. 邻接表的邻接结点采用什么顺序链接?  
书中采用头结点插入,也就是后边的顶点靠前,这个程序采用正序链表,但是需要解决头结点和非头节点的区分
3. DFS 非递归遍历时没有访问到最深处结点?  
设立结点已访问的次序出现问题,开始是在入栈时就设置已访问,这样同一个元素不能多次入栈,正确顺序应该是弹出栈顶元素后判断是否以访问,若未访问则输出元素,然后标记已访问。这样才能实现 DFS。

源程序名称: 1163450201.c

注意: 正文文字为宋体小 4 号, 图中文字为宋体 5 号。行距为多倍行距 1.25。

源程序与此报告打包提交, 压缩包采用学号命名。