实验九　图实验

【实验目的】

（1）掌握图的基本概念及其邻接矩阵、邻接表两种存储结构。

（2）掌握图在两种存储结构上的深度优先遍历、广度优先遍历算法。

【实验背景】

1、 图的存储

邻接矩阵是表示图中各顶点之间关系的矩阵，是图的存储结构之一。

定义 10.14 假设 G = {V，E}是一个有 n 个顶点的图，我们规定各顶点的序号依次为 1，2，3，…… ，n，则 G 的邻接矩阵是一个具有如下定义的 n 阶方阵：

$$A[i,j]=\begin{cases}1, & \text{若} <V_i,V_j> \text{或者} (V_i,V_j)\in \mathrm{E(G)}\\ 0, & \text{反之}\end{cases}$$

对于在边上附有权值的网，可以将以上的定义修正为：

$$A[i,j]=\begin{cases}\mathrm{W_i}, & \text{若} <V_i,V_j> \text{或者} (V_i,V_j)\in \mathrm{E(G)}\\ 0, & \text{反之}\end{cases}$$

其中 Wi 表示<V_i，V_j>弧或(V_i，V_j)边上的权值。

一个图的邻接矩阵存储结构可以用两个数组来表示。其中第一个数组 vexs 是一维数组，用来存储图中顶点的信息；另外一个二维数组 edges，用来存储图中边或弧的信息。邻接矩阵数据类型如下：

```
# define MAX_VERTEX_NUM 100          //顶点的最大个数
typeof struct {                      //定义顶点类型
   int num;                          //顶点序号
   char data;                        //顶点信息
} VERTEX;
typeof struct {                      //定义图的类型
   int n;                            //顶点数目
   int e;                            //边或弧的数目
   VERTEX vexs[MAX_VERTEX_NUM];      //一维数组，存储顶点
   int edges[MAX_VERTEX_NUM] [MAX_VERTEX_NUM];//二维数组，存储边或弧
} MGraph;
```

邻接表是图的链式存储结构。在邻接表存储结构中，图中的每个顶点对应一个单链表，第 i 个单链表中的节点表示依附于顶点 Vi 的边(对于有向图，表示以 Vi 为尾的弧)。

单链表中每个节点有三个域组成，如图 11.1 所示：

中：adjvex 为邻接点域，指示与顶点 Vi 相邻接的顶点在图中的位置(序号)；nextarc 为链域(指针域)，指示依附于顶点 Vi 的下一条边或弧的节点；info 为信息域，存储与边或弧

adjvex	nextarc	info

图 11.1 邻接表的节点结构

相关的信息，如权值。一个顶点的所有相关边的节点通过链域 nextarc 相连接，组成该顶点的单链表。

在每个单链表的前面附设一个表头节点指示对应的顶点。它由两个域组成，结构如图 11.2 所示：

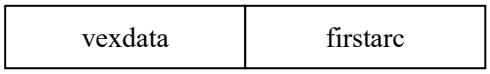

图 11.2 邻接表的表头节点

其中：vexdata 为数据域，存储顶点的信息，如顶点位置；firstarc 为链域，指向单链表中的第一个节点，即依附于顶点 Vi 的第一条边。

这些表头节点通常以顺序结构(一维数组)的方式存储，从而可以从表头中方便地访问任一单链表。

邻接表数据类型(存储结构)描述如下：

```
# define MAX_VERTEX_NUM 100          //顶点的最大个数
typedef struct {
  int adjvex;                        //邻接点的位置
  ARCNODE *nextarc;                        //指向下一个节点
  char info;                         //边的信息
} ARCNODE;                                 //邻接表中的节点类型
typedef struct VEXNODE {
  char vexdata;                      //顶点信息
  ARCNODE *firstarc;                 //指向第一个邻接节点
} VEXNODE，AdjList[MAX_VERTEX_NUM];          //表头节点类型
typedef struct {
  AdjList vextices;
  int vexnum，arcnum;
} ALGraph;                           //邻接表类型
```

2、 图的遍历

从图的任一顶点出发访问图中其余顶点，使每个顶点被访问且仅被访问一次。这一过程称为图的遍历。图的遍历算法通常有两个：深度优先搜索和广度优先搜索。

深度优先搜索(depth-first search 简称 dfs)类似于树的先根遍历，是先根遍历在图的一种推广。

其搜索的过程(算法)如下：

(1)首先访问某一个指定的顶点 V_0。

(2)依次从 v0 的未被访问的邻接点出发作深度遍历。

广度优先搜索(breadth-first search 简称 bfs)类似于按树的层次遍历的过程。其搜索过程(算法)如下：

(1)从图中某个顶点 v0 出发，首先访问 v0。

(2)依次访问 v0 的各个未被访问的邻接点。

(3)分别从这些邻接点（端结点）出发，依次访问它们的各个未被访问的邻接点（新的端结点）。

【实验任务】

1、程序验证

（1）阅读下列程序，指出程序的功能，并通过运行验证之。

```
typedef   struct node{
    int   adjvex;
    struct node   *next;
}edgnode;
typedef   struct {
```

```
    vextype    vertex;
    edgnode    *link;
}vexnode;

vexnode    ga[n];

void    creatdjlist(vexnode    ga[]){
    int    i, j, k;
    edgnode    *s;
    for(i=0; i<n; i++){
      ga[i].vertex=getchar();
        ga[i].link=NULL;
    }
    for(k=0; k<e; k++) {
        scanf("%d%d", &i, &j);
        s=malloc(sizeof(edgnode));
        s->adjvex=j;
        s->next=ga[i].link;
        ga[i].link=s;
        s=malloc(sizeof(edgnode));
        s->adjvex=i;
        s->next=ga[j].link;
        ga[j].link=s;
      }
}
```

（2）在第（1）题的基础上编写广度优先遍历程序，输出遍历序列。

（3）建立无向图的邻接矩阵存储，并分别对建立的无向图进行深度优先遍历和广度优先遍历。

（4）建立有向图的邻接表存储，并分别对建立的有向图进行深度优先遍历和广度优先遍历。

2、算法填空

（1）下列函数 MDFSForest 的功能是，对一个采用邻接矩阵作存储结构的图进行深度优先搜索遍历，输出所得深度优先生成森林中各条边。请在空缺处填入合适内容，使其成为一个完整的算法。

```
#define MaxMun 20    //图的最大顶点数
typedef struct {
    char vexs [MaxNum];     //字符类型的顶点表
    int edges [MaxNum][MaxNum];       //邻接矩阵
    int n, e;                                  //图的顶点数和边数
}MGraph;         //图的邻接矩阵结构描述
typedef enum {FALSE, TRUE} Boolean;
Boolean visited [MaxNum];
 void MDFSTree (MGraph *G, int i);
void MDFSForest (MGraph *G)
{
```

```
    int i, k;
    for (i=0; i <G -> n; i++)
        visited [i] = _____ ;
    for (k = 0; k<G -> n; k++)
      if (!visited [k]) MDFSTree (G,k);
}
void MDFSTree (MGraph *G, int i)
{  int j;
   visited [i]=TRUE;
   for (j=0; j<G -> n; j++)
   {   if ( _____ )
            {   printf ( 〃 <%c, %c> 〃 G -> vexs [i], G -> vexs [j]);
                _____ ;
            }
   }
}
```

（2）完善下述算法，并通过运行来验证：求图中出度为 0 的顶点个数。

```
# define MAX_VERTEX_NUM 100
typedef struct {
   int num；
   char data；
} VERTEX；
typedef   struct {
   int n,   e；
   VERTEX vexs[MAX_VERTEX_NUM]；
   int [MAX_VERTEX_NUM] [MAX_VERTEX_NUM]；
} MGraph；
int    sumzero(MGraph    A ) {
    count = 0;
    for( i=0;    _____ ;   i++)
    {   tag = 0;
        for( j=0;    _____ ;   j++)
          if (A..edges[i][j]!= _____ )
            {   tag = 1;   break; }
        if( tag = = 0)
          count++;
     }
    return   count;
  }
```

3、算法设计

（1）假设以邻接矩阵作为图的存储结构，编写算法判断在给定的有向图中是否存在一个简单有向回路。

提示：判别一个图是否有回路，可以有以下几种方法：

（1）利用深度优先遍历；

（2）拓扑排序。

（2）采用邻接表存储有向图，设计算法判断任意两个顶点间是否存在路径。

（3）设计算法，将一个无向图的邻接矩阵转换为邻接表。

（4）设计算法建一个无向图的邻接表转换为邻接矩阵。

4、实例演练： 医院选址问题

[问题描述]

N 个村庄之间的交通图可以用有向网表示，图中边<vi, vj>上的权值表示从村庄 i 到村庄 j 的道路长度。现在要从这 n 个村庄中选择一个村庄新建一所医院，问这所医院应建在哪个村庄，才能使所有村庄离医院都比较近？

[测试数据]

假设表示 5 个村庄的有向网如图 11.3 所示：

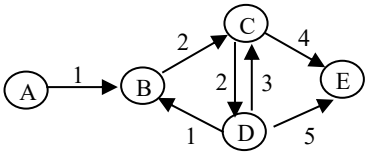

图 11.3 村庄有向网示意图

[基本要求]

① 建立模型，设计存储结构；

② 设计算法完成问题求解；

③ 分析算法的时间复杂度。

[实现提示]

医院选址问题实际是求有向图中心点问题。首先定义顶点的偏心度。

设图 G=(V, E)，对任一顶点 k，称 E(k)= max{d(i,k)}(i∈V)为顶点 k 的偏心度。显然，偏心度最小的顶点即为图 G 的中心点。

医院选址问题算法用伪代码描述如下：

① 对有向网调用 Floyd 算法，求每对顶点间最短路径长度的矩阵；

② 对最短路径长度矩阵的每列求最大值，即得到各顶点的偏心度；

③ 具有最小偏心度的顶点即为所求。

[思维扩展]

图的存储结构和算法的设计需要一定的灵活性和技巧。从医院选址问题的求解过程中，你得到什么启发？