



## 第10周小结

1

### 图的逻辑结构

#### ① 逻辑表示方式

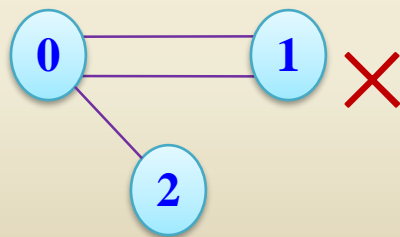
- 图形表示：直接用图表示
- 二元组表示： $G=(V, E)$ ， $V$ 为顶点集， $E$ 为边集

## ② 逻辑特性

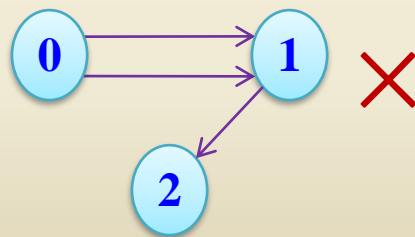
- 顶点之间多对多关系
- 无向关系  $\Rightarrow$  无向图
- 有向关系  $\Rightarrow$  有向图



数据结构中讨论的图是没有多重边的！顶点编号： $0 \sim n-1$



$(0, 1)$  无向边出现两次



$\langle 0, 1 \rangle$  有向边出现两次



若无向图 $G(V, E)$ 中含7个顶点，则保证图 $G$ 在  
任何情况下都是连通的，则需要的边数最少是（ ）。

A. 6

B. 15

C. 16

D. 21

- 对于具有 $n$ 个顶点的无向图，当其中 $n-1$ 个顶点构成一个完全图时，再加上一条边（连接该完全图和另外一个顶点）必然构成一个连通图
- 所以本题中，若 6个顶点构成一个完全图，再加上一条边，这样的图无论如何都是一个连通图
- 最少边数 $=\frac{(n-1)(n-2)}{2}+1=16$



下列关于无向连通图特征的叙述中，正确的是（ ）。

I. 所有顶点的度之和为偶数

II. 边数大于顶点个数减1

III. 至少有一个顶点的度为1

A. 只有I

B. 只有II

C. I和II

D. I和III

- 所有顶点的度之和  $= 2e$ ，为偶数  $\Rightarrow$  I正确。
- 无向连通图中， $e \geq n-1$   $\Rightarrow$  II错误。
- 无向连通图中，可能存在度为1的顶点  $\Rightarrow$  III错误。

A

## 2

# 图的存储结构

### ① 图的两种存储方法

- 邻接矩阵
- 邻接表

## ② 图两种存储方法的特点

以下关于图的存储结构的叙述中正确的是\_\_\_\_\_。

- A. 一个图的邻接矩阵表示唯一，邻接表表示唯一
- B. 一个图的邻接矩阵表示唯一，邻接表表示可能不唯一
- C. 一个图的邻接矩阵表示可能不唯一，邻接表表示唯一
- D. 一个图的邻接矩阵表示可能不唯一，邻接表表示可能不唯一

- 一个图的邻接矩阵表示唯一
- 邻接表表示可能不唯一（一个顶点相邻的所有顶点构成一个单链表，其中相邻顶点的节点顺序可以任意）

**B**





以下关于图的存储结构的叙述中正确的是（ ）。

- A. 邻接矩阵占用的存储空间大小只与图中顶点数有关，而与边数无关
  - B. 邻接矩阵占用的存储空间大小只与图中边数有关，而与顶点数无关
  - C. 邻接表占用的存储空间大小只与图中顶点数有关，而与边数无关
  - D. 邻接表占用的存储空间大小只与图中边数有关，而与顶点数无关
- 
- **无向图**：用邻接矩阵存储时，占用的存储空间大小为 $O(n^2)$ ；用邻接表存储时，占用的存储空间大小为 $O(n+2e)$ 。
  - **有向图**：用邻接矩阵存储时，占用的存储空间大小为 $O(n^2)$ ；用邻接表存储时，占用的存储空间大小为 $O(n+e)$

**A**

# 3

## 图的遍历

### ① 遍历过程

- 某种次序
- 访问所有顶点
- 不重复访问



## ② 常用图遍历方法


- 深度优先遍历 ——— 具有递归性
- 广度优先遍历





假设图采用邻接矩阵表示。设计一个从顶点 $v$ 出发的深度优先遍历算法。

找顶点 $v$ 的相邻顶点 $w$

$w$  

0	1	0	1	1
1	0	1	1	0
0	1	0	1	1
1	1	1	0	1
1	0	1	1	0

算法如下:

```
int visited[MAXV];           //全局变量, 所有元素置初值0
void MDFS(MGraph g, int v)
{   int w;
    printf("%d ", v);        //访问顶点v
    visited[v]=1;             //置访问标记
    for (w=0;w<g.n;w++)      //找顶点v的所有相邻点
        if (g.edges[v][w]!=0 && g.edges[v][w]!=INF && visited[w]==0)
            MDFS(g, w); //找顶点v的未访问过的相邻点w
}
```

### ③ DFS遍历算法应用示例



图采用邻接表作为存储结构。对于一个无向连通图 $G$ ，假设不知道 $n$ 和 $e$ ，设计一个算法判断是否为一棵树。若是树，返回true；否则返回false。

- 若 $G \rightarrow e = G \rightarrow n - 1 \Rightarrow$  树图？但 $G \rightarrow e$ 和 $G \rightarrow n$ 未知！
- 对于无向连通图 $G$ ，采用DFS，访问的顶点数 $vn$ 为 $n$ ，试探的边数 $en$ 恰好为 $2e$ 。

$$\begin{aligned} en/2 &= vn - 1 \\ \text{或者 } en &= 2(vn - 1) \end{aligned}$$

一棵树

```

int visited[MaxSize];
void DFS2(ALGraph *G, int v, int &vn, int &en)
{   ArcNode *p;
    visited[v]=1; vn++;           //遍历过的顶点数增1
    p=G->adjlist[v].firstarc;
    while (p!=NULL)
    {   en++;                     //试探过的边数增1
        if (visited[p->adjvex]==0)
            DFS2(G, p->adjvex, vn, en);
        p=p->nextarc;
    }
}

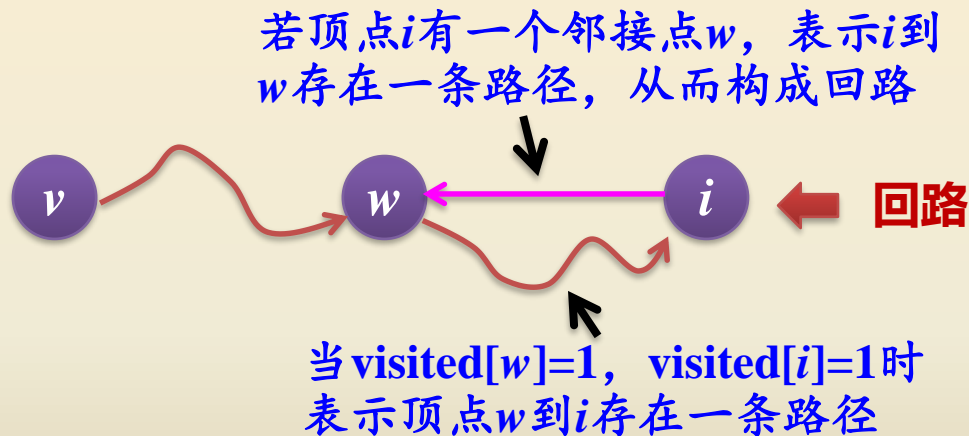
```

**bool GIsTree(ALGraph \*G)   //判断无向图G是否是一棵树**

```
{   int vn=0, en=0, i;  
  for (i=0; i<MaxSize; i++)  
    visited[i]=0;  
  DFS2(G, 0, vn, en);  
  if (en==2*(vn-1))  
    return true;  
  else  
    return false;  
}
```



假设一个连通图采用邻接表作为存储结构。试设计一个算法，判断其中是否存在回路。



**void Cycle(ALGraph \*G, int v, bool &has)**

{ //调用时has置初值false

ArcNode \*p; int w;

visited[v]=1;

//置已访问标记

p=G->adjlist[v].firstarc;

//p指向顶点v的第一个邻接点

while (p!=NULL)

{ w=p->adjvex;

if (visited[w]==0)

//若顶点w未访问，递归访问它

**Cycle(G, w, has);**

else

//又找到了已访问过的顶点说明有回路

has=true;

p=p->nextarc;

//找下一个邻接点

}

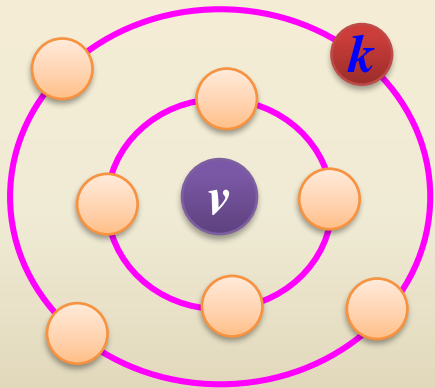
}



#### ④ BFS遍历算法应用示例



假设图 $G$ 采用邻接表存储。设计一个算法，求不带权无向连通图 $G$ 中距离顶点 $v$ 最远的一个顶点。



- 最外圈中的任何一个顶点是最远的顶点
- BFS遍历完毕，队列中最后一个出队且没有相邻访问顶点的顶点 $k$ 属于该圈中的顶点

```
int Maxdist(ALGraph *G, int v)
{   ArcNode *p;
    int Qu[MAXV], front=0, rear=0; //队列及队头、尾指针
    int visited[MAXV], i, j, k;
    for (i=0;i<G->n;i++)           //初始化访问标志数组
        visited[i]=0;
    rear++;Qu[rear]=v;              //顶点v进队
    visited[v]=1;                   //标记v已访问
```

```

while (rear!=front)
{
    front=(front+1)%MAXV;
    k=Qu[front];
    p=G->adjlist[k].firstarc;
    while (p!=NULL)
    {
        j=p->adjvex;
        if (visited[j]==0)
        {
            visited[j]=1;
            rear=(rear+1)%MAXV;Qu[rear]=j;
        }
        p=p->nextarc;
    }
}
return k;
}

```

//顶点出队  
 //找第一个邻接点  
 //所有未访问过的邻接点进队  
 //若j未访问过  
 //将顶点j进队

//找下一个邻接点