图

图的定义：

图G由顶点集V和边集E组成，记为$G=(V,E)$，其中$V(G)$表示图G中顶点的有限非空集；$E(G)$表示图G中顶点之间的关系（边）集合

顶点： vertex
边： edge

1．有向图

若E是有向边（也称弧）的有限集合时，则图G为有向图。弧是顶点的有序对，记为<v，w>，其中v，w是顶点，v称为弧尾，w称为弧头(有箭头的那个)，<v，w>称为从v到w的弧，也称v邻接到w。

图(a)所示的有向图G1可表示为

G1 = (V1，E1)

V1 = {1，2，3}

E1 = {<1，2>，<2，1>，<2，3>}

2．无向图

若E是无向边（简称边）的有限集合时，则图G为无向图。边是顶点的无序对，记为(v，w)或(w，v)。可以说w和v互为邻接点。边(v，w)依附于w和v，或称边(v，w)和v，w相关联。

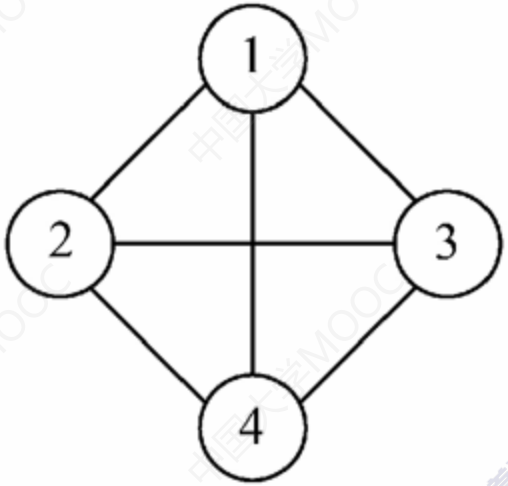
图(b)所示的无向图G2可表示为

G2 = (V2，E2)

V2 = {1，2，3，4}

E2 = {(1，2)，(1，3)，(1，4)，(2，3)，(2，4)，(3，4)}

(a) 有向图G_1



(b) 无向图G_2

图的存储

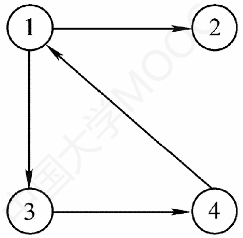
邻接矩阵法与邻接表法，其他存储方法

考研考大题概率约等于零

所谓邻接矩阵存储，是指用一个一维数组存储图中顶点的信息，用一个二维数组存储图中边的信息（即各顶点之间的邻接关系），存储顶点之间邻接关系的二维数组称为邻接矩阵。

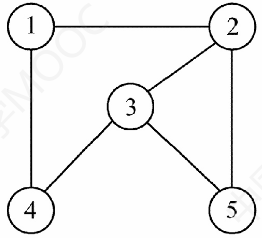
图的邻接矩阵存储结构定义如下：

```
#define MaxVertexNum l00     //顶点数目的最大值
typedef char VertexType;                    //顶点的数据类型
typedef int EdgeType;                       //带权图中边上权值的数据类型
typedef struct{
        VertexType Vex[MaxVertexNum];          //顶点表
        EdgeType Edge[MaxVertexNum][MaxVertexNum]; //邻接矩阵，边表
        int vexnum,arcnum;                  //图的当前顶点数和弧数
}MGraph;
```



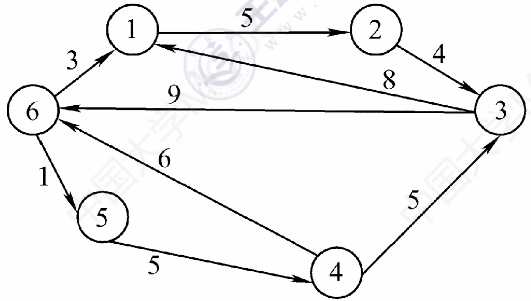
$$A_1=\begin{bmatrix}0&1&1&0\\0&0&0&0\\0&0&0&1\\1&0&0&0\end{bmatrix}$$

(a) 有向图G_1及其邻接矩阵



$$A_2=\begin{bmatrix}0&1&0&1&0\\1&0&1&0&1\\0&1&0&1&1\\1&0&1&0&0\\0&1&1&0&0\end{bmatrix}$$

(b) 无向图G_2及其邻接矩阵



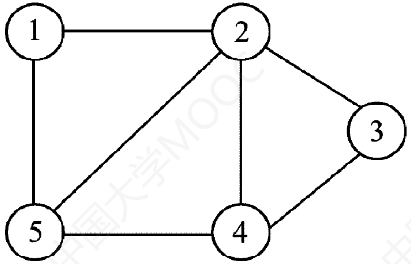
$$A_3=\begin{bmatrix}\infty&5&\infty&\infty&\infty&\infty\\\infty&\infty&4&\infty&\infty&\infty\\8&\infty&\infty&\infty&\infty&9\\\infty&\infty&5&\infty&\infty&6\\\infty&\infty&\infty&5&\infty&\infty\\3&\infty&\infty&\infty&1&\infty\end{bmatrix}$$

(c) 网及其邻接矩阵

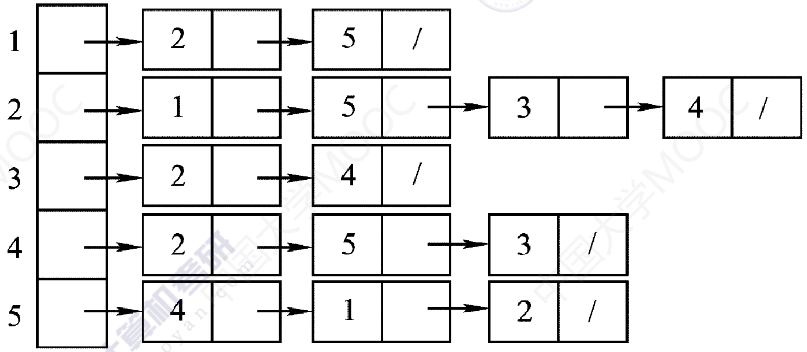
邻接表法

当一个图为稀疏图时，使用邻接矩阵法显然要浪费大量的存储空间，而图的邻接表法结合了顺序存储和链式存储方法，大大减少了这种不必要的浪费。

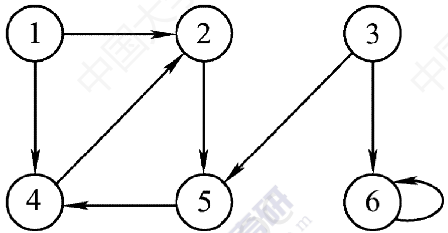
所谓邻接表，是指对图G中的每个顶点vi建立一个单链表。

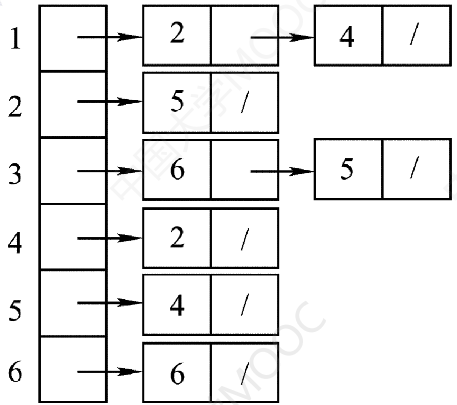


(a)无向图G



(b)图G的邻接表的表示



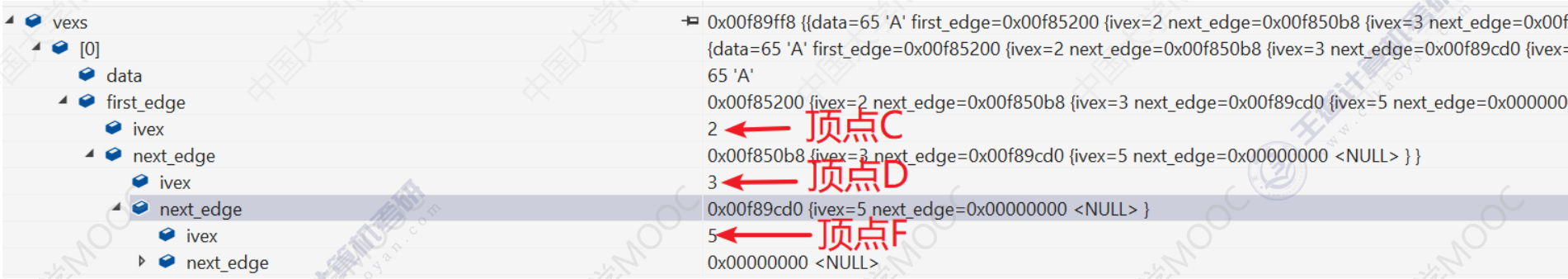


```
typedef struct _ENode  // 邻接表中表对应的链表的顶点
{
    int ivex;                  // 该边所指向的顶点的位置
    struct _ENode *next_edge;  // 指向下一条弧的指针
}ENode, *PENode;

typedef struct _VNode  // 邻接表中表的顶点
{
    char data;             // 顶点信息
    ENode *first_edge;     // 指向第一条依附该顶点的弧
}VNode;

typedef struct _LGraph  // 邻接表
{
    int vexnum;            // 图的顶点的数目
    int edgnum;            // 图的边的数目
    VNode vexs[MAX];
}LGraph;
```

单步调试无向图

通过单步调试查看发现为每一个顶点的相邻结点建立了一条链表

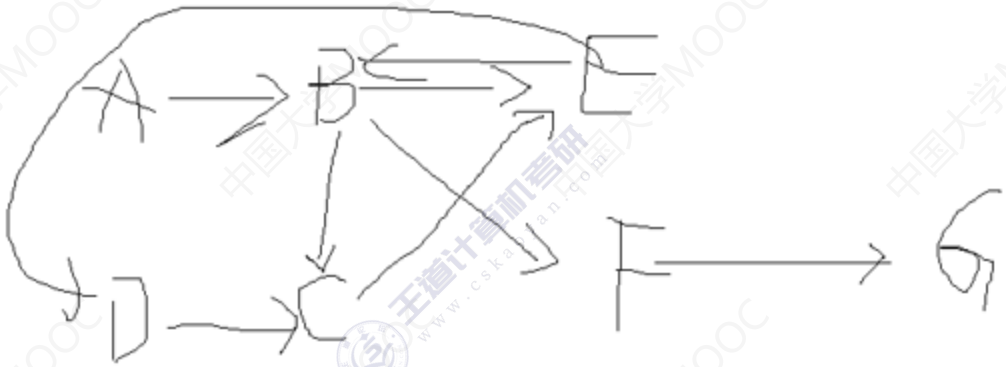
```
char edges[][2] = {
    {'A', 'C'},
    {'A', 'D'},
    {'A', 'F'},
    {'B', 'C'},
    {'C', 'D'},
    {'E', 'G'},
    {'F', 'G'}};
```



有向图的深度优先遍历

基本思想如下：首先访问图中某一起始顶点v，然后由v出发，访问与v邻接且未被访问的任一顶点w_1，再访问与w_1邻接且未被访问的任一顶点w_2……重复上述过程。当不能再继续向下访问时，依次退回到最近被访问的顶点，若它还有邻接顶点未被访问过，则从该点开始继续上述搜索过程，直至图中所有顶点均被访问过为止。

和树的深度优先遍历很像，看下页的图



通过该图对照代码来学习

有向图的广度优先遍历

基本思想是：首先访问起始顶点v，接着由v出发，依次访问v的各个未访问过的邻接顶点$w_1, w_2, \ldots, w_i$，然后依次访问$w_1, w_2, \ldots, w_i$的所有未被访问过的邻接顶点；再从这些访问过的顶点出发，访问它们所有未被访问过的邻接顶点，直至图中所有顶点都被访问过为止