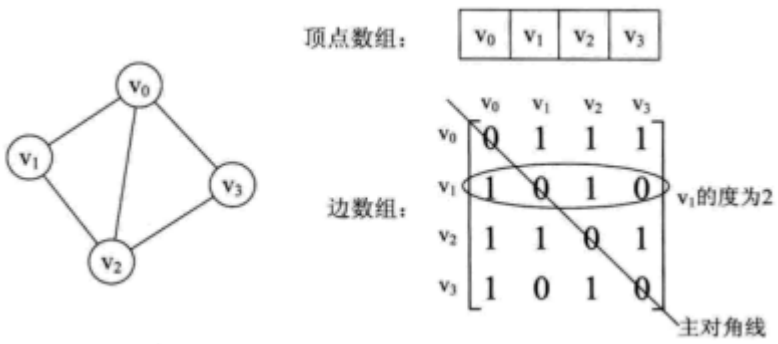
数组表示法

用两个数组分别存储数据元素（顶点）的信息和数据元素之间的关系的信息

```
/**
 * 图的类型
*/
typedef enum GraphKind{
    Digraph,                //有向图
    DirectedNetwork,        //有向网
    Undigraph,              //无向图
    UndirectedNetwork       //无向网
} GraphKind;

/**
 * 邻接矩阵
*/
typedef struct ArcCell {
    //VRType为顶点关系类型;
    //对于无权图，用1或0表示相邻否;
    //对于有权图，则为权值类型；不邻接用无穷表示
    VRType      adjacency;
    InfoType    *info;          //该弧相关信息的指针
} ArcCell, AdjacencyMatrix[MAX][MAX];

/**
 * 图
*/
typedef struct {
    VertexType          verties[MAX];       //顶点向量
    AdjacencyMatrix     arcs;               //邻接矩阵
    int                 vertexNumber;       //当前顶点数
    int                 arcNumber;          //当前弧数
    GraphKind           kind;               //图的种类
} Graph;
```



以二维数组表示有n个顶点的图时，需存放n个顶点信息和n^2个弧信息的存储量；

借助于邻接矩阵容易判断任意两个顶点之间是否有边（或）弧相连；

并容易求得各个顶点得度；

- 对于无向图，顶点vi的度是邻接矩阵中第 i行（或第i列）的元素之和
- 对于有向图，第i行的元素之和为顶点vi的出度，第j列的元素之和为顶点vj的入度

优点

- 容易实现图的操作

缺点

- 空间效率O(n^2)，对于稀疏图来说浪费空间

邻接表（Adjacency list）

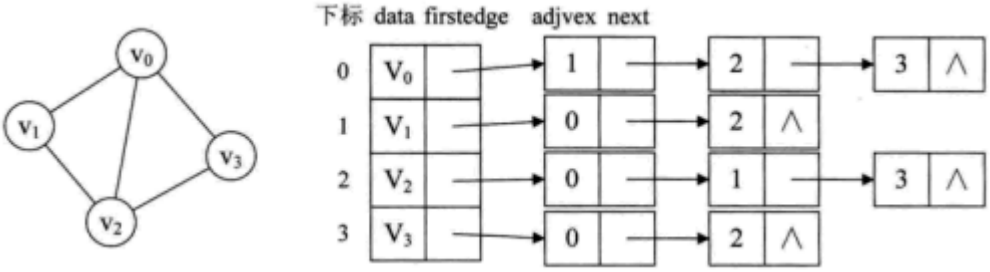
邻接表是图的一种链式存储结构；

在邻接表中，对图中每个顶点建立一个单链表，第i个单链表中的结点表示依附于vi的边（对有向图是以顶点vi为尾的弧）

```
/**
* 边
*/
typedef struct Arc{
    int             adjacencyVertex;    //邻接顶点
    struct Arc      *nextArc;           //指向下一条弧的指针
    InfoType        *info;              //该弧相关信息的指针
} Arc;

/**
* 顶点
*/
typedef struct {
    VertexType      data;               //顶点信息
    Arc             *firstArc;          //边链表；指向第一条依附于该顶点的弧的指针
} Vertex, AdjacencyList[MAX];

/**
* 图
*/
typedef struct {
    AdjacencyList       vertices;       //顶点集合
    int                 vertexNumber;   //当前顶点的数目
    int                 arcNumber;      //当前边的数目
    GraphKind           kind;           //图的种类
} Graph;
```



若无向图有n个顶点和e条边，则它的邻接表需n个头结点和2e个表结点；在边稀疏的情况下，用邻接表表示图比邻接矩阵节省存储空间；

在邻接表容易找到任一顶点的第一个邻接点和下一邻接点，但要判断任意两个顶点（vi和vj）之间是否有边或弧相连，则需要搜索第i个或第j个链表；因此，不及邻接矩阵方便

十字链表（Orthogonal List）

十字链表是有向图的另一种链式存储结构