第八章 图

- 图的基本概念
- 图的存储结构
- 图的遍历
- 最小生成树
- 最短路径
- 拓扑排序
- 关键路径

8. 1 图的基本概念

- **图定义** 图是由顶点集合(vertex)及顶点间的关系集合组成的一种数据结构：

$$Graph = (V, E)$$

其中 $V = \{x \mid x \in$ 某个数据对象$\}$

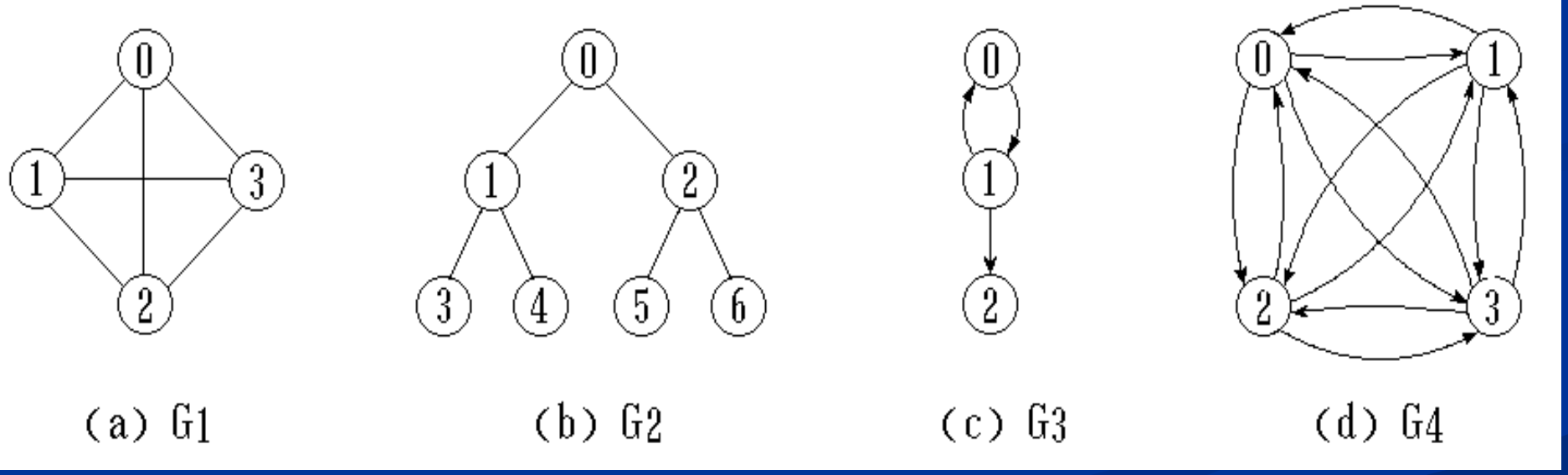
是顶点的有穷非空集合；

$$E = \{(x, y) \mid x, y \in V\}$$

或 $$E = \{<x, y> \mid x, y \in V \ \&\& \ Path(x, y)\}$$

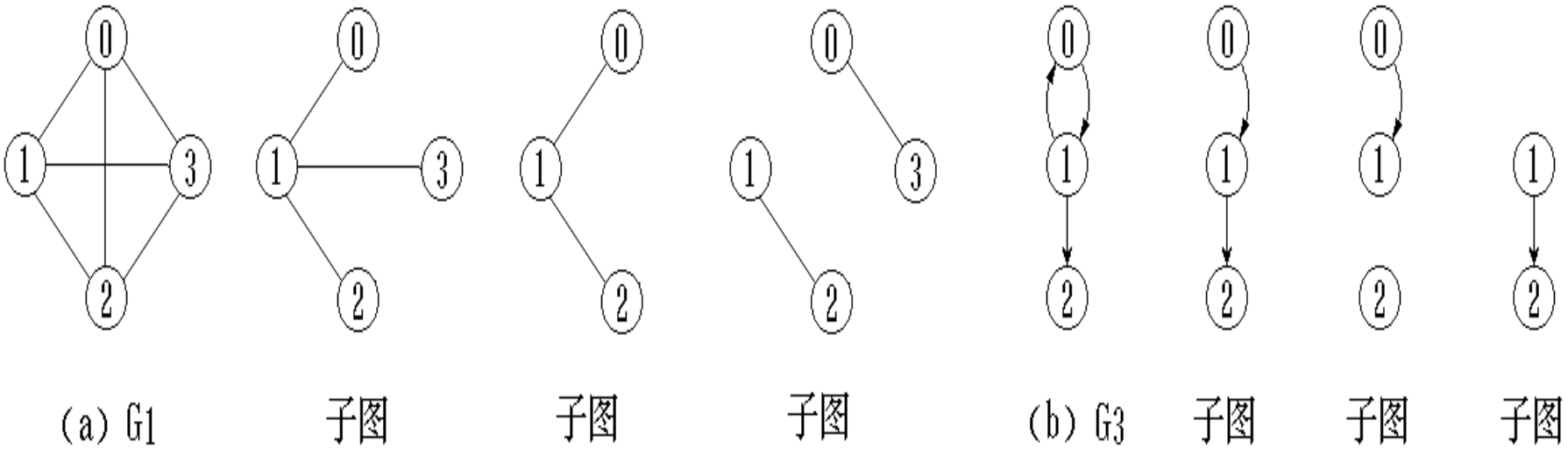
是顶点之间关系的有穷集合，也叫做边(edge)集合。

- **有向图与无向图** 在有向图中，顶点对<x, y>是有序的。在无向图中，顶点对(x, y)是无序的。
- **完全图** 若有 n 个顶点的无向图有 $n(n-1)/2$ 条边，则此图为完全无向图。有 n 个顶点的有向图有 $n(n-1)$ 条边，则此图为完全有向图。



邻接顶点 如果 (u, v) 是 $E(G)$ 中的一条边，则称 u 与 v 互为邻接顶点。

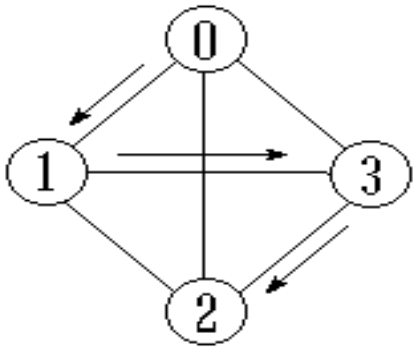
- **权**　某些图的边具有与它相关的数，称之为权。这种带权图叫做**网**。
- **子图**　设有两个图 $G=(V, E)$ 和 $G'=(V', E')$。若 $V'\subseteq V$ 且 $E'\subseteq E$，则称 图G' 是 图G 的子图。



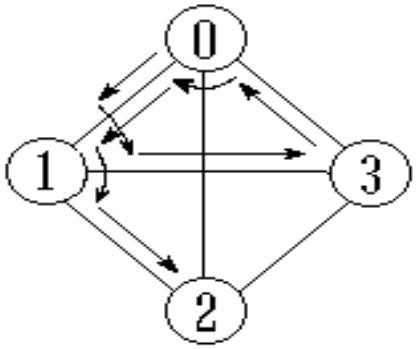
顶点的度　一个顶点v的度是与它相关联的边的条数。记作TD(v)。在有向图中，顶点的度等于该顶点的入度与出度之和。

- **顶点 v 的入度**是以 v 为终点的有向边的条数, 记作 ID(v); **顶点 v 的出度**是以 v 为始点的有向边的条数, 记作 OD(v)。
- **路径**　在图 G = (V, E) 中, 若从顶点 v_i 出发, 沿一些边经过一些顶点 v_{p1}, v_{p2}, …, v_{pm}, 到达顶点v_j。则称顶点序列 (v_i v_{p1} v_{p2} … v_{pm} v_j) 为从顶点v_i 到顶点 v_j 的路径。它经过的边<v_i, v_{p1}>、 <v_{p1}, v_{p2}> 、 … 、 < v_{pm}, v_j >应是属于E的边。
- **路径长度**
  - 非带权图的路径长度是指此路径上边的条数。
  - 带权图的路径长度是指路径上各边的权之和。

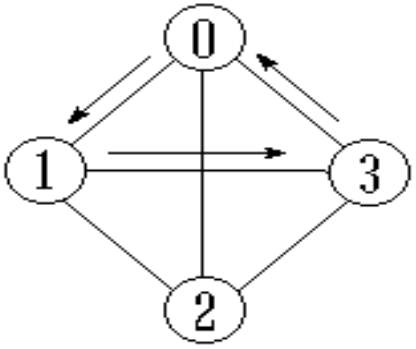
- **简单路径**　若路径上各顶点 $v_1,v_2,\cdots,v_m$ 均不互相重复，则称这样的路径为简单路径。
- **回路**　若路径上第一个顶点 v_1 与最后一个顶点 v_m 重合，则称这样的路径为回路或环。



（a）简单路径



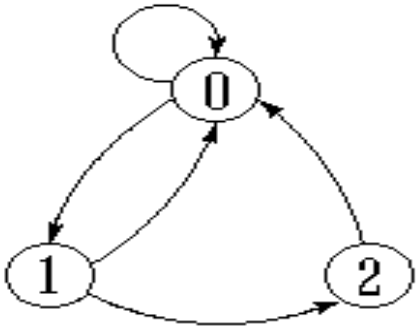
（b）非简单路径



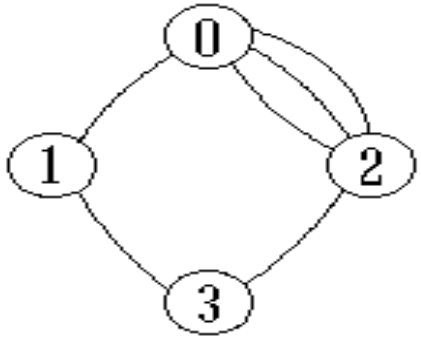
（c）　回路

连通图与连通分量　在无向图中，若从顶点 v_1 到顶点 v_2 有路径，则称顶点 v_1 与 v_2 是连通的。如果图中任意一对顶点都是连通的，则称此图是连通图。非连通图的**极大连通子图**叫做连通分量。

- **强连通图与强连通分量**　在有向图中，若对于每一对顶点v_i和v_j，都存在一条从v_i到v_j和从v_j到v_i的路径，则称此图是强连通图。非强连通图的**极大强连通子图**叫做强连通分量。
- **生成树**　一个连通图的生成树是它的**极小连通子图**，在n个顶点的情形下，有n-1条边。但有向图则可能得到它的由若干有向树组成的生成森林。
- **本章不予讨论的图**：



（a）带自身环的图



（b）多重图

8.2 图的存储结构

一. 图的数组表示法 (邻接矩阵表示法)

基本思想：

- ✓ 图需要存储的信息：顶点和弧。
- ✓ **引入两个数组，一个记录各个顶点信息的顶点表，还有一个表示各个顶点之间关系的邻接矩阵。**

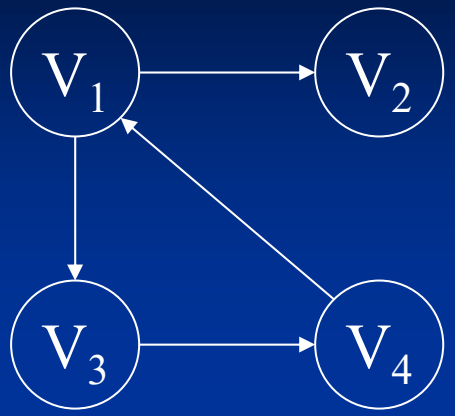
- **设图 A = (*V*, *E*)是一个有 *n* 个顶点的图，则图的邻接矩阵是一个二维数组 *edges*[*n*][*n*]，定义：**

$$edges[i][j] = \begin{cases} 1 & \text{若} <v_i, v_j> \text{或} (v_i, v_j) \in E \\ 0 & \text{反之} \end{cases}$$

下面两个图的邻接矩阵分别为：

edges =

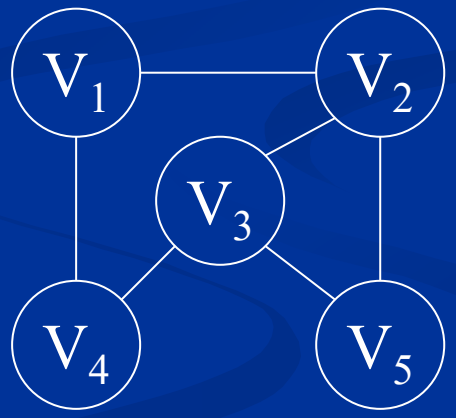
$$\begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$



有向图G1

edges =

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{pmatrix}$$

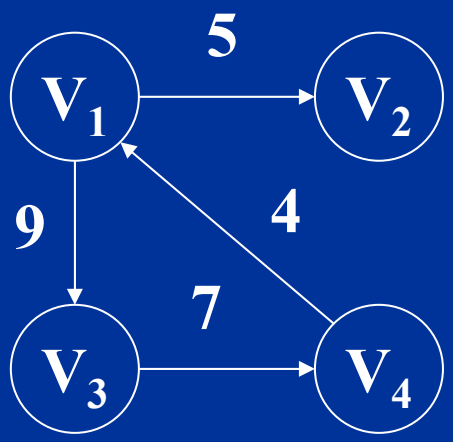


无向图G2

网的邻接矩阵可以定义为：

$$edges[i][j] = \begin{cases} W_{i,j} & \text{若} <v_i, v_j> \text{或} (v_i, v_j) \in E \\ 0\text{或}\infty & \text{反之} \end{cases}$$

有向网G1的邻接矩阵为：



有向图G1

$$edges = \begin{pmatrix} \infty & 5 & 9 & \infty \\ \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & 7 \\ 4 & \infty & \infty & \infty \end{pmatrix}$$

邻接矩阵的特点：

- ✓ 无向图的邻接矩阵一定是一个对称矩阵。
- ✓ 无向图的邻接矩阵的第i行(或第i列)非零元素(或非∞元素)个数为第i个顶点的度$D(v_i)$。
- ✓ 有向图的邻接矩阵的第i行非零元素(或非∞元素)个数为第i个顶点的出度$OD(v_i)$，第i列非零元素(或非∞元素)个数就是第i个顶点的入度$ID(v_i)$。

图的数组(邻接矩阵)存储表示：

```
enum graphType{undigraph, digraph,
undinetwork, dinetwork};

template<class T>
struct EdgeType{
    T head, tail;
     int cost;
     EdgeType(T h, T t, int c)
    {
        head=h; tail=t; cost=c;
    }
};
```

```
template<class T>
class MGraph{
    int vexnum, edgenum;   // 图中的顶点数、边数
    graphType  kind;  //图的类型标记
    vector <vector <int>> edges;  //邻接矩阵
    vector <T> vexs;   // 顶点表
    void DFS(int v, bool* visited);  //连通图的深度优先遍历
public:
    MGraph(graphType t, T v[], int n, int e);
    ~MGraph( ){};   //析构函数
    int vertexNum();  //返回图中的顶点数量
    int edgeNum();  //返回图中的边数量
    void DFSTraverse();  //深度优先遍历图
    void BFSTraverse ();  //广度优先遍历图
};
```

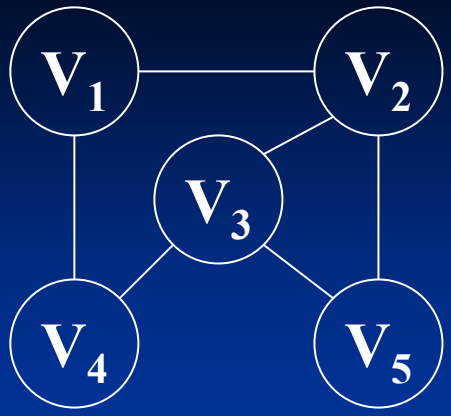
无向网的邻接矩阵建立算法：

```
template<class T>
MGraph<T>::MGraph( graphType t, T v[], int n, int e) {
    vexnum=n;
    edgenum=e;
    kind=t;
    vexs.resize(vexnum);
    edges.resize(vexnum);
    for(i=0;i<vexnum;i++)      //初始化顶点表
        vexs[i]=v[i];
    for(i=0;i<vexnum;i++)  //根据图中的顶点数预置邻接矩阵的大小
        edges[i].resize(vexnum);
    for(i=0;i<n; i++)     //初始化邻接矩阵
        for(j=0;j<n;j++)
            edges[i][j]= INFINITY;
```

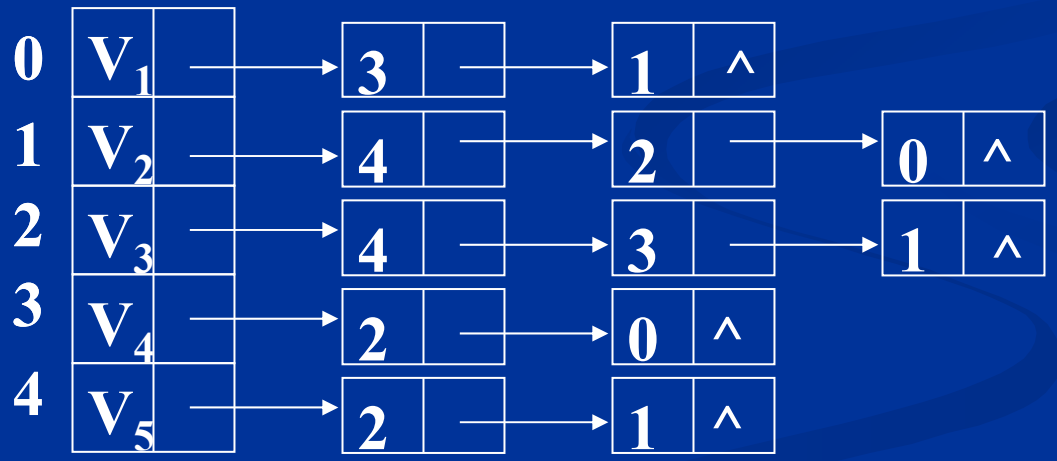
```
        for(i=0;i<e;i++) {        //依次输入所有的边的信息
            cin>>va>>vb;
            cin>>w;
            edges[va][vb]=edges[vb][va]= w;
        }
}
```

二. 邻接表 (Adjacency List)

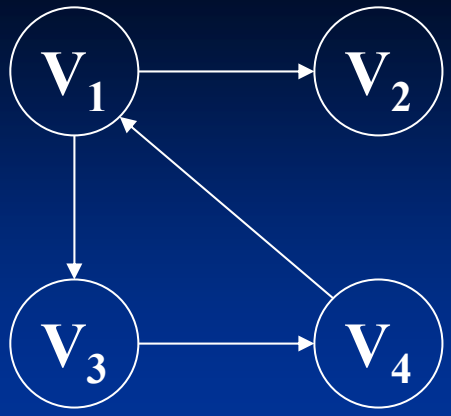
- 基本思想：
  - 如果对图中的每个顶点都建立一个单链表（边表）来存储所有依附于该顶点的弧或边，就可以把图中所有已有的弧或边的信息保存下来。
  - 对于图中所有顶点还是使用一个一维数组来存放。
- 在邻接表表示法中，对于顶点单元i，需要存放的内容有顶点信息以及指向依附于该顶点的所有的边组成的单链表（边表）。
- 对于边表中的每个结点，需要存放该边指向的顶点的位置（也就是该边依附的另一个顶点的位置）和指向下一条边的指针。



无向图G1

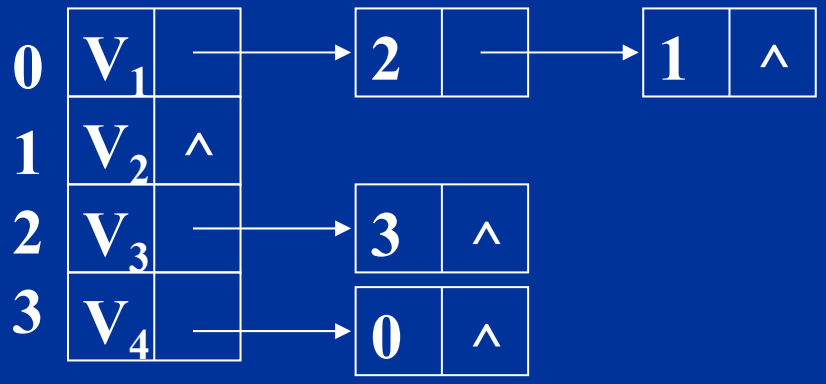


无向图G2的邻接表



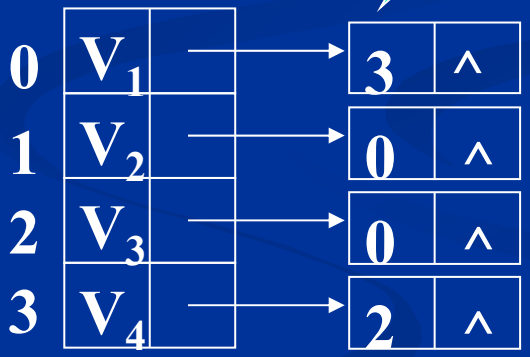
有向图G2

出边表



有向图G2的邻接表

入边表



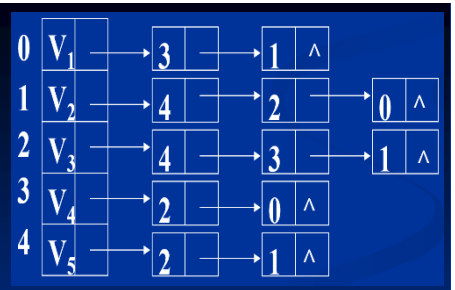
有向图G2的逆邻接表

图的邻接表存储表示：

```
 struct EdgeNode{      // 边表的结点结构类型
        int adjvex;        // 该边的终点位置
        EdgeNode* nextedge;   // 指向下一条边的指针
};

template<class T>
struct VexNode{   //顶点表的元素结构类型
        T data;
        EdgeNode* firstedge;
};
```

```
template<class T>
class ALGraph{
    int vexnum, edgenum;   //图中的顶点数、边数
    vector<VexNode<T>>  adjlist;    //顶点表
    graphType  kind;  //图的类型标记
    void DFS(int v, bool* visited);  //连通图的深度优先遍历
public:
    ALGraph(graphType t, T vexs[], int n, int e);  //构造函数
    ~ALGraph( );   //析构函数
    EdgeNode* firstEdge(int v);  //返回第v个顶点对应的边表的头指针
    void DFSTraverse();  //深度优先遍历图
    void BFSTraverse ();  //广度优先遍历图
     ……
};
```



☞ （无向图）邻接表的生成算法：

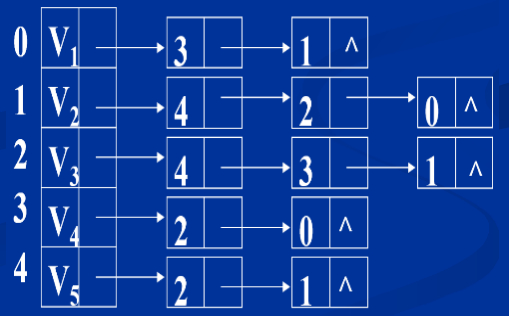
算法思想：

（1）依次读入图的各顶点数据并将其存入到头结点数组中，并将表头结点数组的链域均置“空”；

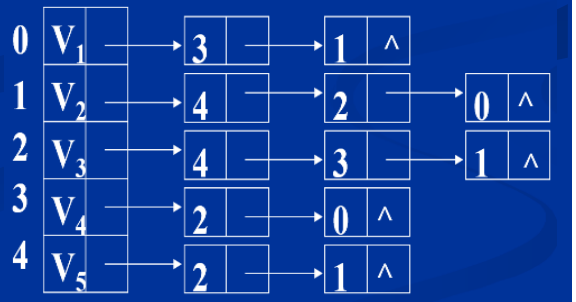
（2）逐个输入表示边的顶点序号（i，j）；约定i>0&&j>0是图的有效顶点对。

- 对于每一个有效的顶点对（i，j），动态生成两个结点，它们的顶点域 分别为j，i，并分别插入到第i个和第j个链表（为简便起见，插入的位置为第一个结点之前）中。

```
template<class T>
ALGraph<T>::ALGraph(GraphType t, T vexs[], int n, int e){
        EdgeNode* p;
        vexnum=n;          //确定图的顶点个数和边数
        edgenum=e;
        kind=t;
        adjlist.resize(vexnum);
        for(i=0;i< vexnum;++i)        //初始化顶点表
        {
                adjlist[i].data= vexs[i];
                adjlist[i].firstedge=0;
        }
```



```
for(j=0;j< edgenum; j++ )    //依次输入所有的边的信息
{
        cin>>va>>vb;      // 输入一条边邻接的两个顶点的序号
        p=new EdgeNode;  //产生第一个表结点
        p->adjvex=vb;
        p->nextedge=adjlist[va].firstedge; // 插在表头
        adjlist[va].firstedge=p;
        p=new EdgeNode;   // 产生第二个表结点
        p->adjvex=va;
        p->nextedge=adjlist[vb].firstedge; // 插在表头
        adjlist[vb].firstedge=p;
}
}
```



$$\begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{pmatrix}$$

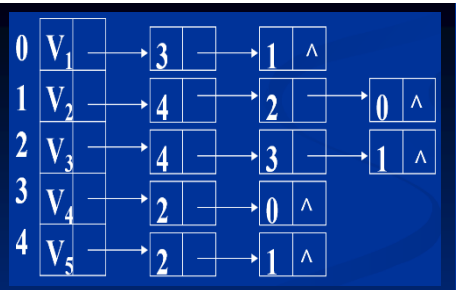
邻接矩阵表示法的优点：

- ✓ 邻接矩阵表示图，很容易确定图中任意两个顶点之间是否有边相连。
- ✓ 容易判断两个顶点之间是否有长度为m的路径相连。

邻接矩阵表示法的缺点：

- 邻接矩阵占用的存储单元个数只与图中的结点个数有关，而与边的数目无关。一个n个结点的图，若其边数比 n^2 少得多，那么邻接矩阵中会存在大量的无用单元。

邻接表的优缺点：



优点：

- ✓ 在边稀疏的情况下，节省存储空间
- ✓ 易于确定图中任一顶点的度数和它的所有邻接点。

缺点：

- ➢ 判定任意两个顶点之间是否有边或弧不方便。