数据结构与算法

Data Structures and Algorithms

第四部分 图

数据结构考查内容

一、线性表
（一）线性表的基本概念
（二）线性表的实现
（三）线性表的应用

二、栈、队列和数组
（一）栈和队列的基本概念
（二）栈和队列的顺序存储结构
（三）栈和队列的链式存储结构
（四）多维数组的存储
（五）特殊矩阵的压缩存储
（六）栈、队列和数组的应用

三、树与二叉树
（一）树的基本概念
（二）二叉树
（三）树、森林
（四）树与二叉树的应用

四、图
（一）图的基本概念
（二）图的存储及基本操作
（三）图的遍历
（四）图的基本应用

数据结构

五、查找
（一）查找的基本概念
（二）顺序查找法
（三）分块查找法
（四）折半查找法
（五）B树及其基本操作，B+树的基本概念
（六）散列（Hash）表
（七）字符串模式匹配
（八）查找算法分析及应用

六、排序
（一）排序的基本概念
（二）插入排序
（三）起泡排序
（四）简单选择排序
（五）希尔排序
（六）快速排序
（七）堆排序
（八）二路归并排序
（九）基数排序
（十）外部排序

算 法

教学要求

- 了解图的定义及相关的术语，掌握图的逻辑结构及其特点；
- 了解图的存储方法，重点掌握图的邻接矩阵和邻接表存储结构；
- 掌握图的遍历方法，重点掌握图的两种遍历算法的实现；
- 了解图型结构的应用，关节点与双连通性求解算法、强连通性判定与强连通分支求解算法，重点掌握最小生成树算法、最短路径算法、拓扑排序和关键路径算法的基本思想、算法原理和实现过程。

考纲内容

（一）图的基本概念

（二）图的存储结构及基本操作

　　邻接矩阵法；邻接表法；邻接多重表；十字链表

（三）图的遍历

　　深度优先搜索；广度优先搜索

（四）图的应用

　　最小（代价）生成树；最短路径；拓扑排序；关键路径

图

- 图的基本概念
- 图的存储结构：邻接矩阵法；邻接表法；邻接多重表；十字链表
- 图的两种遍历方式：深度优先搜索；广度优先搜索
- 图的应用
  - 最小生成树：**Prim**算法；**Kruskal**算法；
  - 最短路径：**Dijkstra**算法；**Floyd**算法；
  - 拓扑排序：**AOV**网；
  - 关键路径：**AOE**网。

知识点与以往课程对比

- 图的类型定义（集合与图论）
- 图的存储表示（集合与图论讲过矩阵的存储）
- 图的深度优先搜索遍历
- 图的广度优先搜索遍历
- 无向网的最小生成树（集合与图论）
- 最短路径（集合与图论讲过 单源最短路径）
- 拓扑排序
- 关键路径

本章难点

- 最小生成树的算法（集合与图论）；
- 拓扑排序的算法;
- 关键路径算法;
- 求最短路径的Dijkstra算法和Floyed算法。

（集合与图论、算法设计与分析）

先提几个问题：

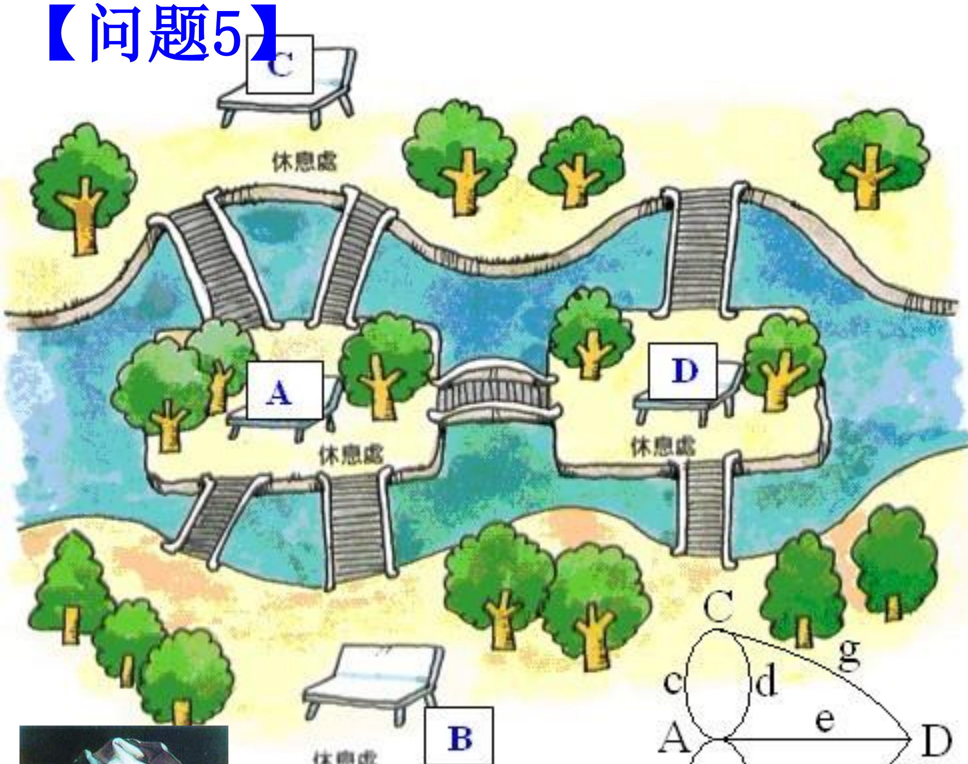
【问题1】由于大道路网的维护成本高，需选择停止维护一些道路，但要保证所有村庄之间都有路到达，即使路线并不如以前短，但要使得总的维护费用最少。

【问题2】给一堆格式为A ＜ B 的关系式，判断有没有一个可以排列的顺序。当一个升序序列确定时，输出处理到第几个关系式和排好序的升序序列；当不确定时，输出也不确定；当矛盾时，输出发现矛盾时处理到第几个关系。

【问题3】有一个长方形的房间，铺设了红色或黑色的方型瓷砖。一名男子站在一个黑色的瓷砖上，从一个瓷砖，他可以转移到四个相邻瓷砖，他只能移动在黑瓷砖上，不能站在红瓷砖上，通过走过的黑瓷砖计算黑瓷砖的片数。

【问题4】计算国际象棋中骑士从一个指定的位置到达目的位置的最少步数。因为每一次都有8种走法，要把可行的走法记录下来，直到走到终点为止。输出最少的步数。

【问题5】



哥尼斯堡是东普鲁士的首都，今俄罗斯加里宁格勒市，普莱格尔河横贯其中。

十八世纪在这条河上建有七座桥，将河中间的两个岛和河岸联结起来。人们闲暇时经常在这上边散步，有人提出：能不能每座桥都只走一遍，最后又回到原来的位置？

——哥尼斯堡城七桥问题

1736年，大数学家欧拉首先把这个问题简化，他把两座小岛和河的两岸分别看作四个点，而把七座桥看作这四个点之间的连线，A、B、C，D表示陆地，形成了著名的——欧拉图。

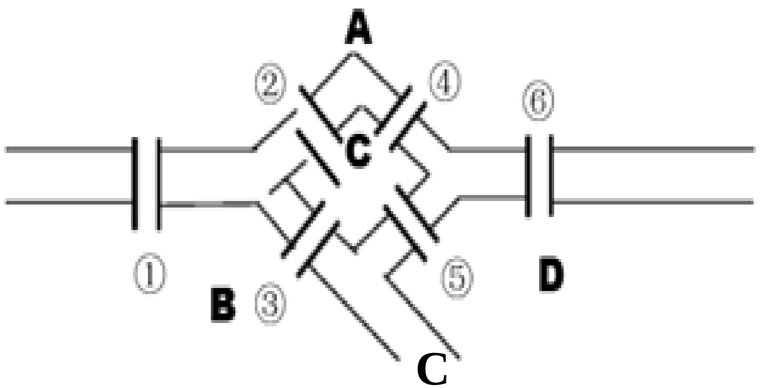
一笔画问题：

（1）由偶点组成的连通图，可以一笔画成。任一偶点为起点，一定能以这个点为终点画完此图；

（2）只有两个奇点的连通图（其余都为偶点），可以一笔画成。把一个奇点为起点，另一个奇点为终点；

（3）其他情况的图都不能一笔画出。(奇点数除以二便可算出此图需几笔画成)。

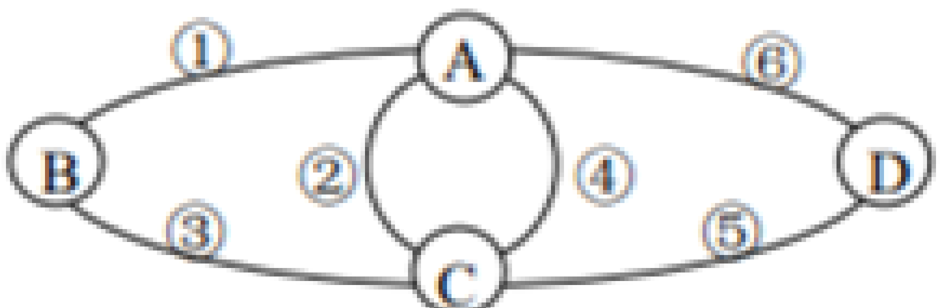
【问题6】简化的格尼斯堡城问题



设在4地（A，B，C，D）之间架设有6座桥，要求从某一地出发，经过每座桥恰巧一次，最后仍回到原地。

（1）此问题有解的条件是什么？

（2）描述与求解此问题有关的数据结构并编写一个算法，找出满足要求的一条回路。



主要内容

4.1	基本术语（图论）	4.7	强联通图
4.2	图的表示（图论讲部分）	4.8	拓扑分类
4.3	图的搜索算法	*4.9	关键路径
4.4	图与树的联系	4.10	最短路径（图论讲部分）
4.5	无向图的双连通性	实验4	图的建立及应用（待定）
*4.6	有向图的搜索	*4.11	每一对顶点间的最短路径

4.1 基本定义/术语

【定义】一个图G=（V，E）是一个由非空的有限集 V 和一个边集E所组成的。若E中的每条边都是顶点的有序对（v ，w），就说该图是有向图（Directed Graph，Digraph）。若E中的每条边是两个不同顶点无序对，就说该图是无向图，其边仍表示成（v, w）。

【ADT】 Graph G = (V , R)

数据对象V：V是具有相同特性的数据元素的集合，称为顶点集。

数据关系R：

R = { VR }

VR = { <v , w >|v, w ∈V,且P(v,w),<v, w>表示从v到w的弧，谓词P(v, w)定义了弧 <v, w>的意义或信息 }

ADT 操作

ADT Graph {

数据对象 V:V 是具有相同特性的数据元素的集合,称为顶点集。

数据关系 R:

R = {VR}

VR = {<v,w>| v,w∈V 且 P(v,w),<v,w>表示从 v 到 w 的弧,

谓词 P(v,w)定义了弧<v,w>的意义或信息 }

基本操作 P:

CreateGraph(&G,V,VR);

初始条件:V 是图的顶点集,VR 是图中弧的集合。

操作结果:按 V 和 VR 的定义构造图 G。

DestroyGraph(&G);

初始条件:图 G 存在。

操作结果:销毁图 G。

LocateVex(G, u);

初始条件:图 G 存在,u 和 G 中顶点有相同特征。

操作结果:若 G 中存在顶点 u,则返回该顶点在图中位置;否则返回其他信息。

GetVex(G, v);

初始条件:图 G 存在,v 是 G 中某个顶点。

操作结果:返回 v 的值。

PutVex(&G, v, value);

初始条件:图 G 存在,v 是 G 中某个顶点。

操作结果:对 v 赋值 value。

FirstAdjVex(G, v);
　初始条件:图 G 存在,v 是 G 中某个顶点。
　操作结果:返回 v 的第一个邻接顶点。若顶点在 G 中没有邻接顶点,则返回“空”。

NextAdjVex(G, v, w);
　初始条件:图 G 存在,v 是 G 中某个顶点,w 是 v 的邻接顶点。
　操作结果:返回 v 的(相对于 w 的)下一个邻接顶点。若 w 是 v 的最后一个邻接点,则返回“空”。

InsertVex(&G, v);
　初始条件:图 G 存在,v 和图中顶点有相同特征。
　操作结果:在图 G 中增添新顶点 v。

DeleteVex(&G, v);
　初始条件:图 G 存在,v 是 G 中某个顶点。
　操作结果:删除 G 中顶点 v 及其相关的弧。

InsertArc(&G, v, w);
　初始条件:图 G 存在,v 和 w 是 G 中两个顶点。
　操作结果:在 G 中增添弧<v,w>,若 G 是无向的,则还增添对称弧<w,v>。

DeleteArc(&G, v, w);
　初始条件:图 G 存在,v 和 w 是 G 中两个顶点。
　操作结果:在 G 中删除弧<v,w>,若 G 是无向的,则还删除对称弧<w,v>。

DFSTraverse(G, Visit());
　初始条件:图 G 存在,Visit 是顶点的应用函数。
　操作结果:对图进行深度优先遍历。在遍历过程中对每个顶点调用函数 Visit 一次且仅一次。一旦 visit()失败,则操作失败。

BFSTraverse(G, Visit());
　初始条件:图 G 存在,Visit 是顶点的应用函数。
　操作结果:对图进行广度优先遍历。在遍历过程中对每个顶点调用函数 Visit 一次且仅一次。一旦 visit()失败,则操作失败。

}**ADT** Graph

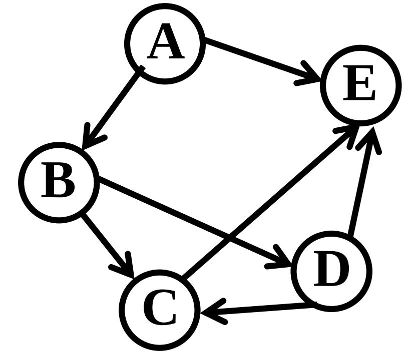
int First Adj Vex (G, v)
//返回值为图G中与顶点v邻接的第一个邻接点，0为没有邻接点

int Next Adj Vex (G, v, w)
//返回值为图G中与顶点v邻接的w之后的邻接点，0为无下一个邻接点

术语：	顶点	弧/边	邻接/相邻	依附
	路径（路）	简单路径	回路	带标号的图（网）
	连通	连通图	强连通图	连通分量
	完全图	稀疏图	稠密图	子图
	度	入度	出度	生成树

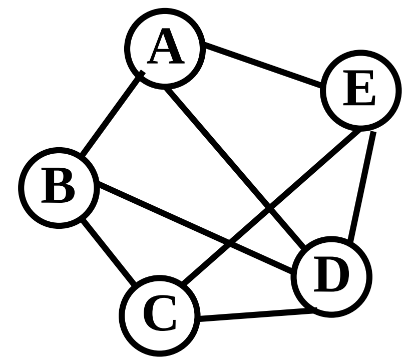
有向图

- ✓ 弧是顶点的有序对，记为<u,w> ∈{VR}
- ✓ <v,w>表示从顶点v到顶点w的一条弧
- ✓ 顶点v为弧尾，顶点w为弧头
- ✓ 由顶点集和弧集构成的图称作有向图



无向图

- ✓ 边是顶点的无序对
- ✓ <u,w> ∈{VR}，则必有<w,u> ∈{VR}
- ✓ <v,w>表示从顶点v到顶点w之间存在一条边
- ✓ 由顶点集和边集构成的图称作无向图



网、子图

- ✓ 弧或边带权的图分别称作有向网或无向网
- ✓ 设 图G＝(V,{VR}) 和 图G'＝(V', {VR'}), 且 V'⊆V, VR'⊆VR，则称G'为G的子图

完全图、稀疏图、稠密图

假设图中有n个顶点，e条边，则

- ✓ 含有 e=n(n-1)/2 条边的无向图称作完全图
- ✓ 含有 e=n(n-1) 条弧的有向图称作有向完全图
- ✓ 若边或弧的个数e<nlog2n，则称作稀疏图，否则称作稠密图

邻接点、度、入度、出度

假若顶点v和顶点w之间存在一条边，则

- ✓ 顶点v和w互为邻接点(相邻)，边(v,w)和顶点v和w相关联
- ✓ 和顶点v关联的边的数目定义为边的度

对有向图来说，

- ✓ 以顶点v为弧尾（起点）的弧的数目定义为顶点的出度
- ✓ 以顶点v为弧头（终点）的弧的数目定义为顶点的入度

度(TD) = 出度(OD) + 入度(ID)

路径、路径长度、简单路径、简单回路

设图G=(V, {VR})中一个顶点序列$\{u=v_{i,0}, v_{i,1},\dots,v_{i,m}=w\}$中，$(v_{i,j-1},v_{i,j})\in VR$, $1<=j<=m$, 则

- ✓ 顶点u到顶点w之间存在一条**路径**
- ✓ 路径上边的数目称作**路径长度**
- ✓ 若序列中的顶点不重复出现，则称作**简单路径**
- ✓ 若u=w，则称这条路径为**回路**或**简单回路**

连通图、连通分量、强连通图、强连通分量

- ✓ 若图G中任意两个顶点之间都有路径相通，则称作此图为**连通图**；
- ✓ 若无向图为非连通图，则图中<u>各个极大连通子图</u>称作此图的**连通分量**。
- ✓ 对有向图，若任意两个顶点之间都存在一条有向路径，则称此有向图为**强连通图**。有向图的极大强连通子图称作它的**强连通分量**。

生成树

- ✓ 假设一个连通图有n个顶点和e条边，其中n-1条边和n个顶点构成一个<u>极小连通子图</u>，称该极小连通子图为此连通图的**生成树**。
- ✓ 对非连通图，由各个连通分量的生成树的集合为非连通图的**生成森林**。

例：下面关于图的表述中，正确的是（III）
I. 回路是简单路径
II. 存储稀疏图，用邻接矩阵比邻接表更省空间
III. 无向图的连通分量是指无向图中的极大连通子图

例：下面关于图的表述中，正确的是（ III ）
I. 假设图G={V,{E}},顶点集 V'∈V，边集E' ∈E，则V'和E'构成的G'是G的子图
II. 强连通有向图的任意顶点到其他所有顶点都有弧
III. 极大连通子图是无向图的连通分量，极小连通子图既要保持图连通又要边最少　提示：极大连通子图的极大要求该连通子图包含其所有边

例：下面关于无向连通图的特性表述中，正确的是（ I ）
I. 所有顶点度之和为偶数；II. 边数大于顶点数减一；III. 至少有一个顶点度为1　提示：II. 生成树不满足；III. 所有顶点构成一个回路，不满足。

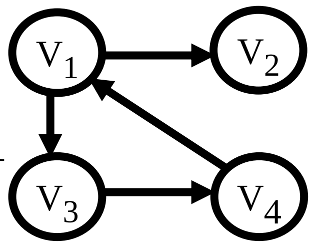
例：若无向图 G = (V, E)中含有7个顶点，要保证图G在任何情况下都是连通的，则需要的边数最小是（C）。
A. 6　B. 15　C. 16　D. 21
提示：任何情况的极端情况，6个顶点是完全无向图，加一条边。

例：一个有28条边的非连通无向图至少有（C） 个顶点。
A. 7　B. 8　C. 9　D. 10

4.2 图的表示

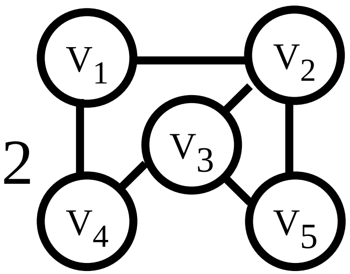
1、图的顺序存储——邻接矩阵

G1

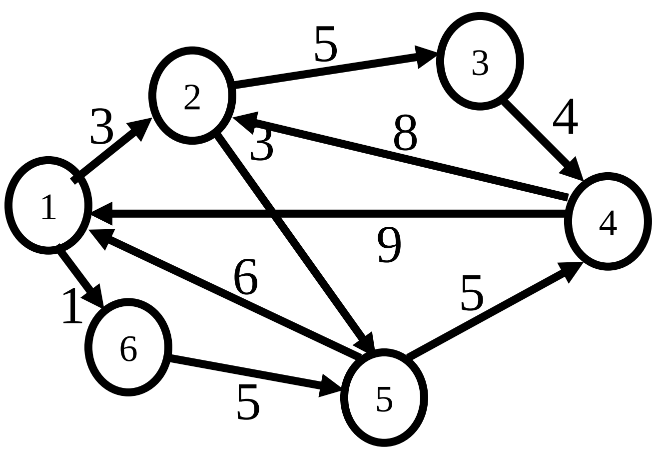


$$
\text{G1.arcs}\quad
\begin{array}{c c}
 & \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\
\begin{array}{c} v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} &
\begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}
\end{array}
$$

G2



$$
\text{G2.arcs}\quad
\begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{pmatrix}
$$



$$
\begin{pmatrix}
\infty & 3 & \infty & \infty & \infty & 1 \\
\infty & \infty & 5 & \infty & 3 & \infty \\
\infty & \infty & \infty & 4 & \infty & \infty \\
9 & 8 & \infty & \infty & \infty & \infty \\
6 & \infty & \infty & 5 & \infty & \infty \\
\infty & \infty & \infty & \infty & 5 & \infty
\end{pmatrix}
$$

设图G = (V, E)，V = { 0， 1， …， n-1 }则表示G的邻接矩阵 A 是其元素按下式定义的n×n矩阵:

$$A[i][j] = \begin{cases} 1 & 若（i, j）\in E \\ 0 & 若（i, j）\notin E \end{cases}$$

网的邻接矩阵可定义为: $$A[i][j] = \begin{cases} w_{ij} & 若（i, j）\in E \\ \infty & 若（i, j）\notin E \end{cases}$$

$$TD(v_i) = \sum_{j=0}^{n-1} A[i][j] = \sum_{j=0}^{n-1} A[j][i]$$ (n :顶点个数，无向图)

$$TD(v_i)=OD(v_i)+ID(v_i) = \sum_{j=0}^{n-1} A[i][j] + \sum_{i=0}^{n-1} A[i][j]$$ (n : 顶点个数，有向图)

图的存储至少存储两个内容：顶点数据和顶点间的关系。

```
#define   INFINITY   INT_MAX
#define   MAX_VERTEX_NUM   20
Typedef   enum { DG, DN, AG, AN }  GraphKind ;
Typedef   struct  ArcCell {
              VRType        adj ; // 顶点的邻接关系
              InfoType      *info ; //弧相关信息的指针
 }  ArcCell  ,  AdjMatrix[ MAX_VERTEX_NUM][MAX_VERTEX_NUM] ;

Typedef   struct  {
             VertexType    vex[MAX_VERTEX_NUM] ;
             AdjMatrix      arcs ;
             int            vexnum , arcnum ;
             GraphKind      kind;
}  Mgraph ;
```

【例4-1】图类型变量：MGraph G ;

顶点个数：	G.vexnum；
弧/边的个数：	G.arcnum；
图的类型：	G.kind = (DG,DN,AG,AN)；
顶点 i 信息：	G.vex[i]；
顶点 i 和顶点 j 邻接关系：	G.arcs[i][j].adj；
弧/边附加信息：	G.arcs[i][j].info->；

如何建立一个简单的邻接矩阵？

邻接矩阵表示法中图的描述

```
#define  n  6      /*图的顶点数*/
#define  e  8      /*图的边数*/
typedef char vexType;  /*顶点的数据类型*/
typedef float adjType; /*权值类型*/
typedef struct
{ vexType vexs[n];
  adjType arcs[n][n];
} graph;
```

$$
\begin{array}{c|c}
 & \begin{matrix} \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{d} & \mathbf{e} \end{matrix} \\
\begin{matrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \\ \mathbf{d} \\ \mathbf{e} \end{matrix} & \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{pmatrix}
\end{array}
$$

图的操作FirstAdjVex()和 NextAdjVex()的实现
存储结构：邻接矩阵

```
int FirstAdjVex(MGraph G,VertexType v)
{ //返回值为图G中与顶点v邻接的第一个邻接点，0为没有邻接点
     VertexType  w=0;
     while((w<G.vexnum) && !G.arcs[v][w].adj)   w++;
     if((w<G.vexnum)&&G.arcs[v][w])    return(w);
     else    return(0);
}
```

```
int NextAdjVex(MGraph G,VertexType v,VertexType w)
{ //返回值为图G中与顶点v邻接的w之后的邻接点，0为无下一个邻接点
      w = w+1;
      while((w<G.vexnum) && !G.arcs[v][w].adj)    w++;
      if((w<G.vexnum)&&G.arcs[v][w])     return(w);
      else    return(0);
}
```

图的邻接矩阵存储表示法特点：

（1）无向图的邻接矩阵一定是一个对称矩阵，可以采用矩阵的压缩存储方式实现。

（2）无向图的第i行（或第i列）的非零元素（或非无穷元素）的个数对应的是该顶点i的度。

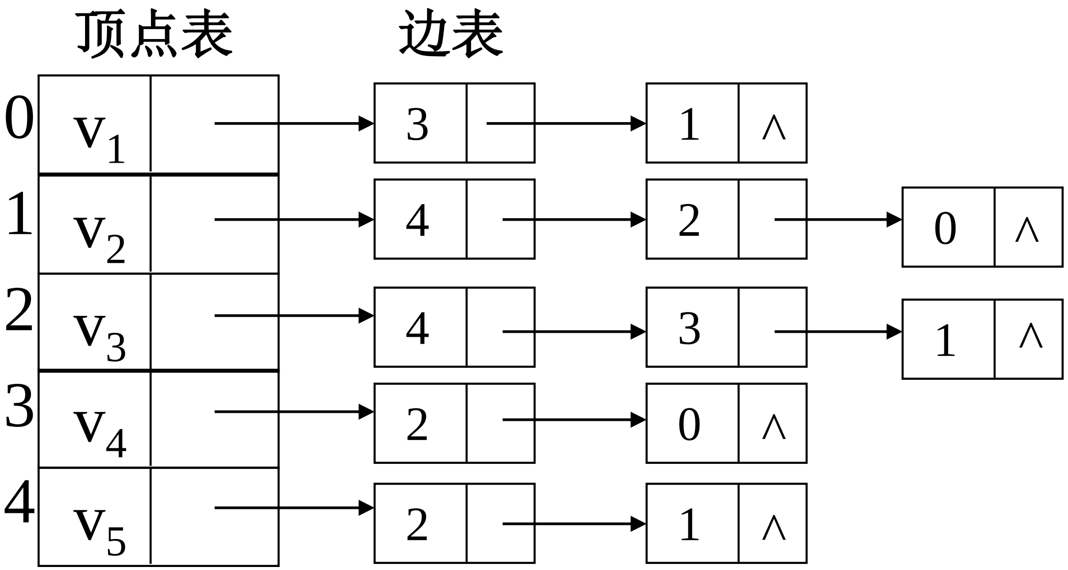
（3）有向图的第i行（第i列）非零元素（或非无穷元素）的个数对应的是该顶点i的出度（入度）。

（4）判断两顶点v、u是否为邻接点：只需判二维数组对应分量是否为1；

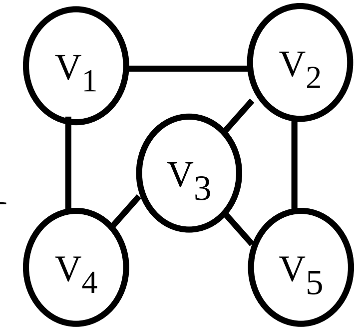
（5）顶点不变，在图中增加、删除边：只需对二维数组对应分量赋值1或清0；

（6）邻接矩阵的局限性：很容易确定任意两顶点是否有边，但比较难确定图中有多少边。

2、图的链式存储——邻接表（Adjacency List）



G1



无向图G1邻接表

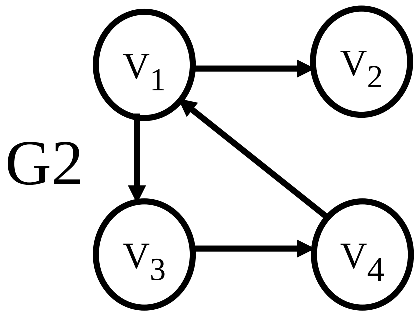
无向图邻接表中第i个顶点的度和图的度：

顶点的度：第i个顶点的边链表结点个数之和；

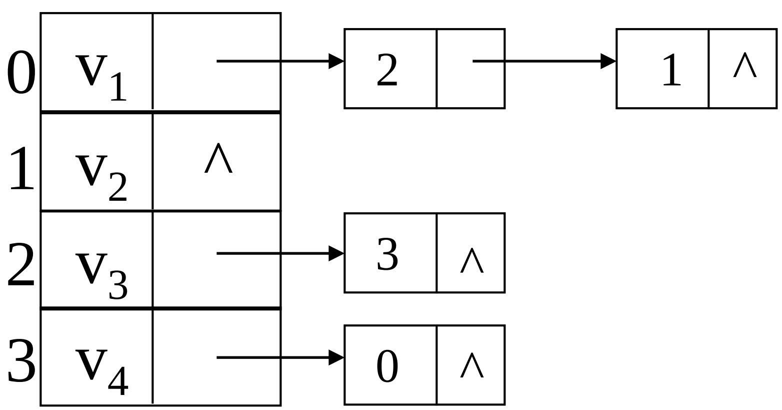
图的总度数：所有边链表结点个数之和；

图的边数：所有边链表结点个数之和的一半。

2、图的链式存储——邻接表（Adjacency List）

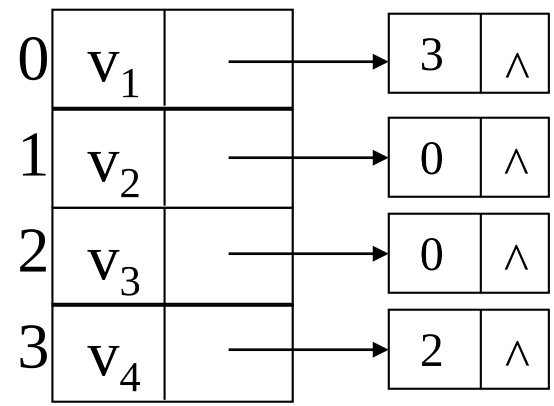


➢ 有向图的邻接表（出度）
- 顶点：用一维数组存储（按编号顺序）
- 以同一顶点为起点的弧：用线性链表存储



有向图G2邻接表

+



G2的逆邻接表

出度：第i个顶点的边链表结点个数之和。

图的边数：所有边链表结点个数之和。

2、图的链式存储（续）

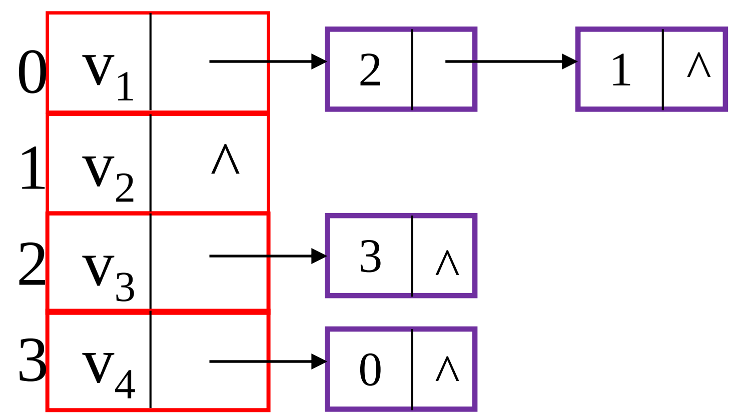
```
#define  MAX_VERTEX_NUM  20
Typedef  struct  ArcNode  {
           int                  adjvex ;//位置
           struct  ArcNode   *nextarc ;
           InfoType          *info ;
}  ArcNode ;

Typedef  struct  Vnode  {
           VertexType     data ;
           ArcNode       *firstarc ;
}  Vnode, AdjList[MAX_VERTEX_NUM] ;

Typedef  struct {
           AdjList    vertices ;
           Int          vexnum ;
           Int          kind ;
}  ALGraph ;
```

表结点（边）

Adjvex	nextarc	info

头结点(顶点)

data	firstarc



【例4-2】图类型变量：ALGraph G ;

顶点个数： G.vexnum;

图的类型： G.kind = (DG,DN,AG,AN);

顶点 i 信息： G.vertices[i].data;

顶点 i 的第一个邻接点：

G.vertices[i].firstarc->adjvex; // int

G.vertices[G.vertices[i].firstarc->adjvex].data;

G.vertices[i].firstarc->info;

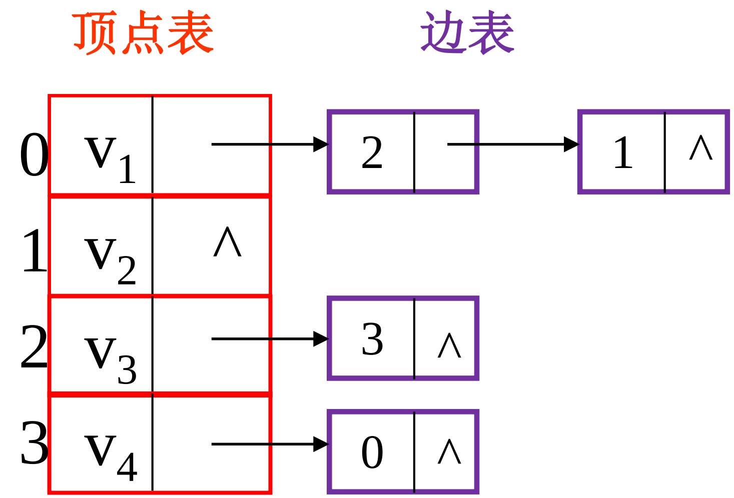
顶点 i 的第二个邻接点：

G.vertices[i].firstarc->nextarc->adjvex;

如何建立一个简单的邻接表？

邻接表的形式说明和建立算法

```
typedef struct      /*顶点表结点定义*/
{VertexType data;
  edgenode *link;
} vexnode;
vexnode ga[n];
typedef struct node /*边表结点定义*/
{ int adjvex; //该弧所指向的顶点的位置
  struct node *next;
} edgenode;
```



图的操作FirstAdjVex()和 NextAdjVex()的实现

存储结构：邻接表

```
int FirstAdjVex(ALGraph G,VertexType v)
{     //返回值为图G中与顶点v邻接的第一个邻接点，0为没有邻接点
      if(G.vertices[v].firstarc)     return(G.vertices[v].firstarc->adjvex);
      else     return(0);
}
```

```
int NextAdjVex(ALGraph G,VertexType v,VertexType w)
{     //返回值为图G中与顶点v邻接的w之后的邻接点，0为无下一个邻接
点
      ArcNode *p;
      p=G.vertices[v].firstarc;
      while(p!=NULL&&p->adjvex!=w)   p=p->nextarc;
      if(p)   return(0);
      else
            if(p->nextarc)     return(p->nextarc->adjvex);
            else     return(0);
}
```

图的邻接表存储表示法特点：

（1）无向图的边数等于邻接表中边结点数的一半，有向图的弧数等于邻接表中边结点数。

（2）存储的空间复杂度：无向图：O(|V|+2|E|)；有向图：O(|V|+|E|)；

（3）邻接表中任意顶点可以很容易找到其所有邻接边，复杂度与边表长有关；但是，如果要确定是否有边存在时，效率较低。

（4）容易找到任一顶点的第一个邻接点。

（5）无向图的邻接表中第i个顶点的度为第i个链表中结点的个数；有向图的邻接表表示法中，一个顶点的出度即为邻接表的结点个数；而入度要么遍历整个邻接表，要么采用逆邻接表法。

（6）图的邻接表表示并不唯一，它与边结点的次序有关，可根据情况调整，但通常采用头插法实现。

补充1：有向图的十字链表(Orthogonal List)表示

邻接表+逆邻接表

弧结点结构

tailvex	headvex	hlink	tlink	info

头结点（顶点结点）结构

data	firstin	firstout

tailvex：尾域,指示弧尾顶点在图中的位置

headvex:头域,指示弧头顶点在图中的位置

hlink：链域,指向弧头相同的下一条弧

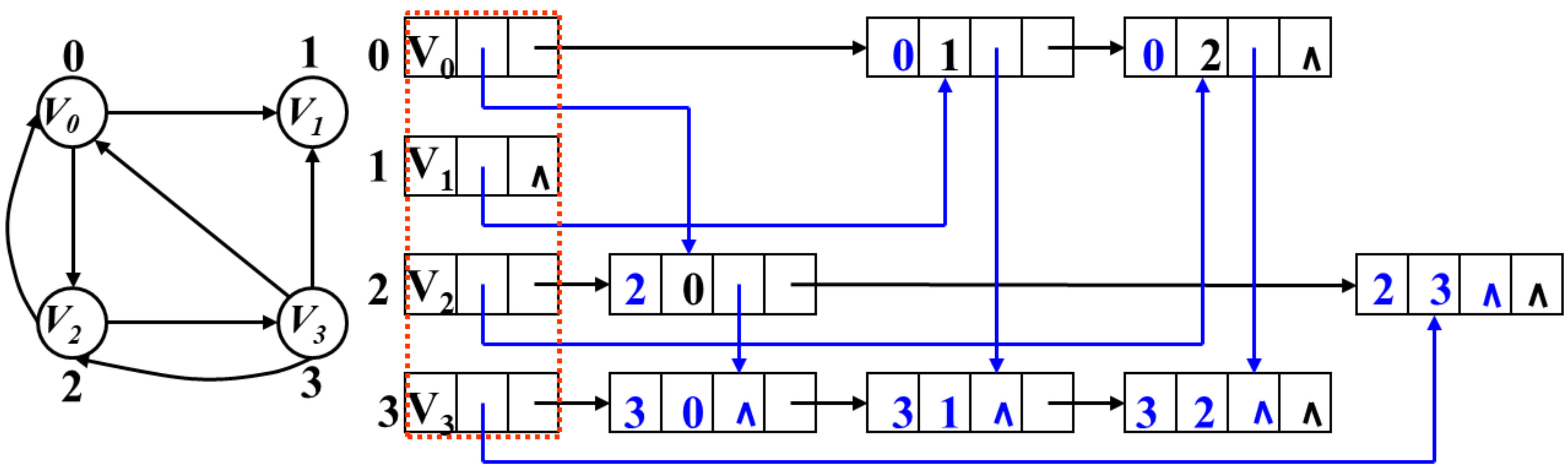
tlink：链域，指向弧尾相同的下一条弧

info：数据域,指向该弧的相关信息

data：数据域，存储和顶点相关的信息，如顶点名称

firstin：链域，指向以该顶点为弧头的第一个弧结点

firstout：链域，指向以该顶点为弧尾的第一个弧结点

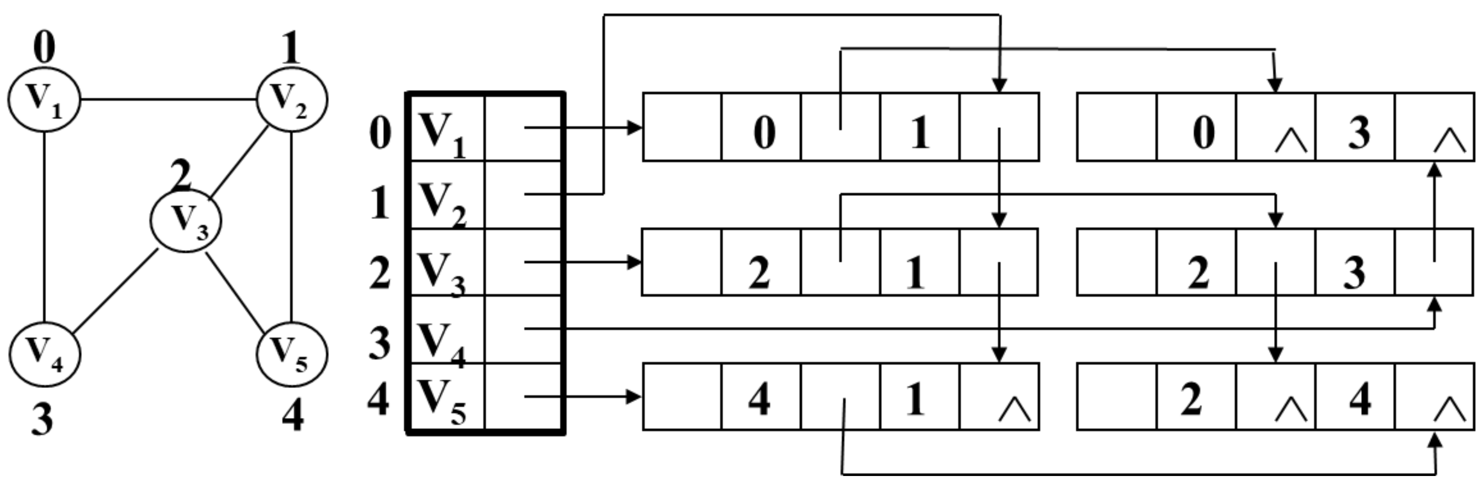


```
#define MAX_VERTEX_NUM   20
typedef struct ArcBox {
        int  tailvex, headvex;                    //该弧的尾和头顶点的位置
        struct ArcBox * hlink, * tlink;  //分别为弧头相同和弧尾相同的弧的链域
        InfoType  info;                           //该弧相关信息的指针
 } ArcBox;

typedef struct VexNode {
      VertexType  data;
      ArcBox  * firstin, * firstout;      //分别指向该顶点第一条入弧和出弧
} VexNode;

typedef struct {
        VexNode  xlist[MAX_VERTEX_NUM];            //表头向量
        int  vexnum, arcnum;       //有向图的当前顶点数和弧数
} OLGraph;
```

补充2：无向图的邻接多重表(Adjacency Multilist)表示

邻接多重表，是对无向图的邻接矩阵的一种压缩表示

- 这种结构在边的操作上会方便，如对已访问的边做标记，或要删除图中某条边，都需找到表示同一条边的两个结点
- 邻接多重表的结构与十字链表类似。在邻接多重表中，所有依附于同一顶点的边串联在同一链表中，由于每条边依附两个顶点，则每个边结点同时链接在两个链表中。



边表的结点结构

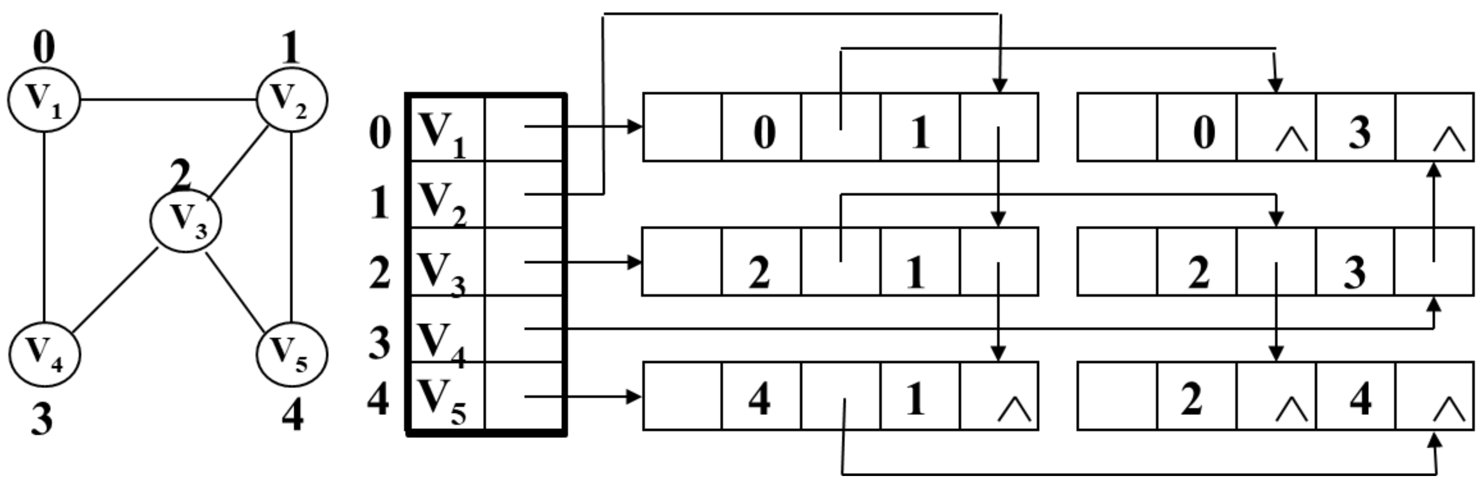
mark	ivex	ilink	jvex	jlink	info

顶点表的结点结构

data	firstedge

mark：标志域，用以标记该条边是否被搜索过
ivex和**jvex**：为该边依附的两个顶点在图中的位置
ilink：链域，指向下一条依附于顶点ivex的边
jlink：链域，指向下一条依附于顶点jvex的边
info：数据域，指向和边相关的各种信息的指针域

Data：数据域，存储和该顶点相关的信息
firstedge：链域，指示第一条依附于该顶点的边



```
#define MAX_VERTEX_NUM   20
typedef emnu {unvisited, visited} VisitIf;
typedef struct EBox {
    VisitIf   mark;              //边访问标记
    int  ivex, jvex;             //该边依附的两个顶点的位置
    struct EBox  * ilink, * jlink;       //分别指向依附这两个顶点的下一条边
    InfoType   *info;                    //该边信息指针
} EBox;

typedef struct VexBox {
    VertexType   data;
    EBox    * firstedge;           //指向第一条依附于该顶点的边
} VexBox;

typedef struct {
    VexBox  adjmulist[MAX_VERTEX_NUM];
     int  vexnum, edgenum;//无向图的当前顶点数和边数
} AMLGraph;
```

例：设图的邻接矩阵右侧邻接矩阵所示，各顶点的度依次是（ C ）

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

A. 1, 2, 1, 2　　B. 2, 2, 1, 1　　C. 3, 4, 2, 3　　D. 4, 4, 2, 2

例：图的存储结构表述中，正确的是（ **B** ）

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，邻接表表示不唯一

C. 一个图的邻接矩阵表示不唯一，邻接表表示唯一

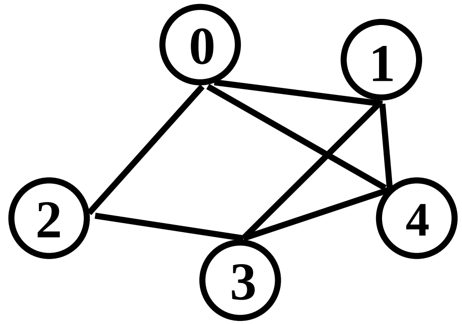
D. 一个图的邻接矩阵和链接表均表示不唯一

例：**n**个顶点的无向图的邻接表最多有（　）个边表结点。

A. n^2　　B. n(n-1)　　C. n(n+1)　　D. n(n-1)/2

提示：完全图，最多有**n(n-1)/2**边，每条边出现两次。

思考题 已知含有5个顶点的图G如下所示：



请回答下面问题：

1）写出图G的邻接矩阵A（行、列下标从0开始）；

2）求A^2，矩阵A^2中位于0行3列元素值的含义是什么？

3）若已知具有n（n>=2）个顶点的图的邻接矩阵为B，则B^m（2=<m<=n）中非零元素的含义是什么？