教学内容

1、图的基本概念；

2、图的存储结构（邻接矩阵、邻接表、十字链表）；

3、图的遍历（深度优先搜索、广度优先搜索）；

4、最小生成树（kruskul算法、prim算法）；

5、最短路径（dijkstra算法、floyd算法）；

6、AOV网络与拓扑排序；

7、AOE网络与关键路径。

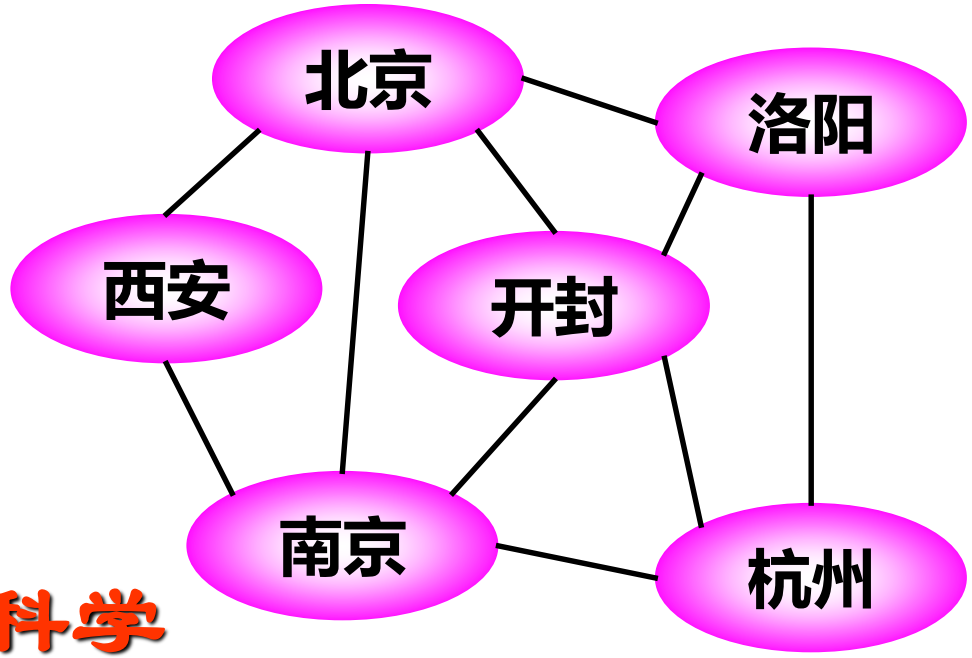
图（Graph）是一种非线性结构。

图的特点
- 顶点之间的关系是任意的
- 图中任意两个顶点之间都可能相关
- 顶点的前驱和后继个数无限制

多对多

图的应用
- 语言学
- 逻辑学
- 物理
- 化学
- 电信工程
- 数学

计算机科学



7.1 图的定义和术语

- 定义：

图是一种：
数据元素间存在多对多关系的数据结构
加上一组基本操作构成的抽象数据类型。

ADT Graph{

数据对象：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 = {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：

- 定义：

图 (Graph) 是一种复杂的非线性数据结构，由顶点集合及顶点间的关系（也称弧或边）集合组成。可以表示为： $G=(V, VR)$

其中 V 是顶点的有穷非空集合； VR 是顶点之间关系的有穷集合，也叫做弧或边集合。弧是顶点的有序对，边是顶点的无序对。

基本操作：

{结构初始化}

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。

{销毁结构}

DestroyGraph(&G);

初始条件：图 G 存在。

操作结果：销毁图 G。

{引用型操作}

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 中存在和 u 相同的顶点，则返回该顶点在图中的位置；否则返回

GetVex(G, v);

初始条件：图 G 存在

操作结果：返回 v 的值

FirstAdjVex(G, v);

初始条件：图 G 存在，v 是 G 中某

操作结果：返回 v 的第一个邻接点。若该顶点在 G 中没有邻接点，则返回“空”。

顶点在图

的是，在图

点之间自然形

若<v,w>∈G，则称 w 为 v 的邻接点，若(v,w)∈G，则称 w 和 v 互为邻接点。

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）下一个邻接点。若 w 是 v 的最后一个邻接点，则返回“空”。

{加工型操作}

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。

若 v 有多个邻接点，则在图的存储结构建立之后，其邻接点之间的相对次序也就自然形成了。

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 是顶点的访问函数。

操作结果：对图 G 进行深度优先遍历。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 是顶点的访问函数。

操作结果：对图 G 进行广度优先遍历。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

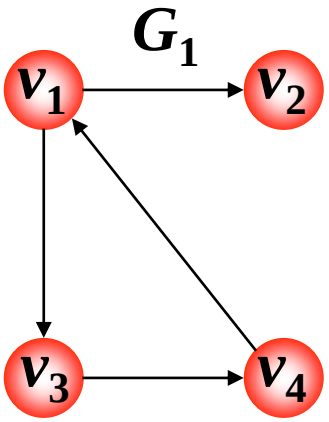
- 基本术语：

顶点： 图中的数据元素。

弧： 若 $<v, w> \in VR$，则 $<v, w>$ 表示从 v 到 w 的一条弧，且称 v 为**弧尾**，称 w 为**弧头**，此时的图称为**有向图**。

$G_1 = (V_1, A_1)$　　$V_1 = \{v_1, v_2, v_3, v_4\}$

$A_1 = \{< v_1, v_2>, < v_1, v_3>, < v_3, v_4>, < v_4, v_1>\}$

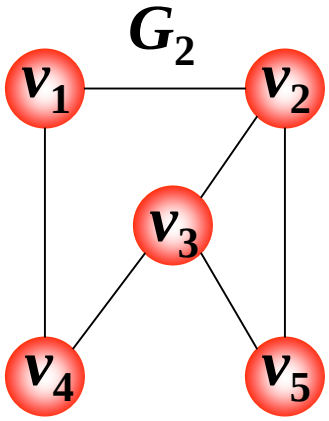


有向图

边：若 $<v, w> \in VR$ 必有 $<w, v> \in VR$，则以无序对 (v, w) 代表这两个有序对，表示 v 和 w 之间的一条边，此时的图称为**无向图**。

$G_2 = (V_2, E_2)$　　$V_2 = \{v_1, v_2, v_3, v_4, v_5\}$

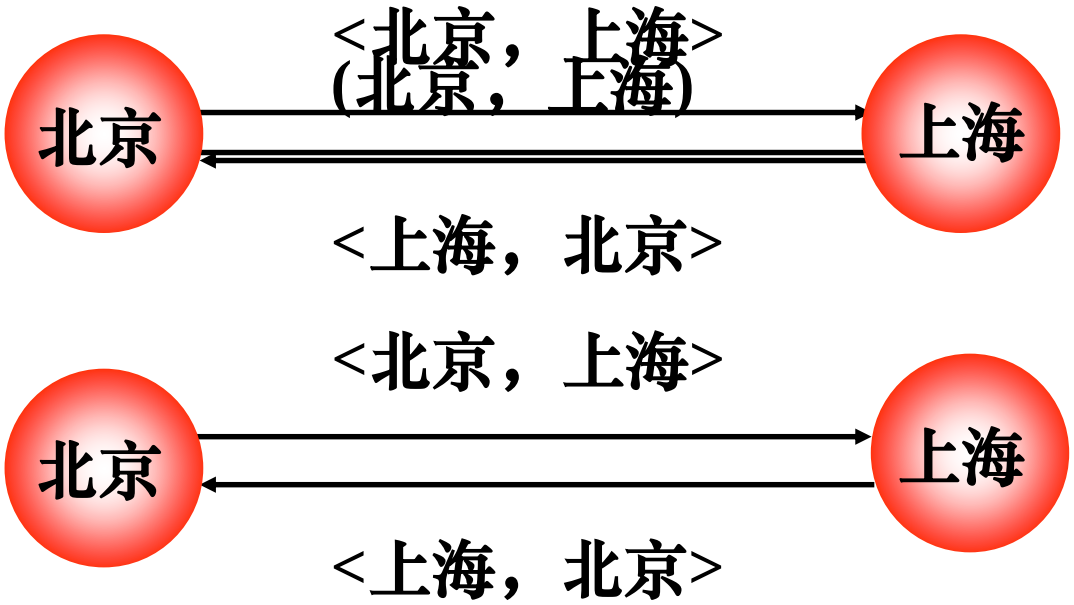
$E_2 = \{(v_1, v_2), (v_1, v_4), (v_2, v_3), (v_2, v_5), (v_3, v_4), (v_3, v_5)\}$



无向图

例：两个城市 A 和 B ，如果 A 和 B 之间的连线的涵义是表示两个城市的距离，则<A, B> 和 <B, A> 是相同的，用 (A, B) 表示。

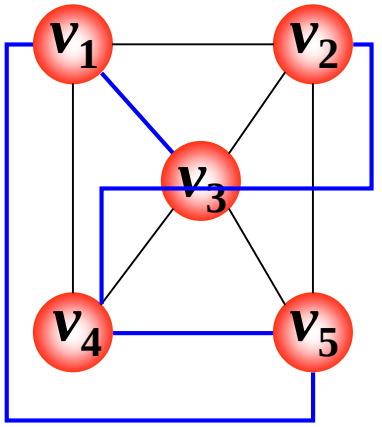
如果 A 和 B 之间的连线的涵义是表示两城市之间人口流动的情况，则 <A, B> 和 <B, A> 是不同的。



无向图中边的取值范围：$0 \le e \le n(n-1)/2$。

（用 n 表示图中顶点数目，用 e 表示边的数目。且不考虑顶点到其自身的边。）

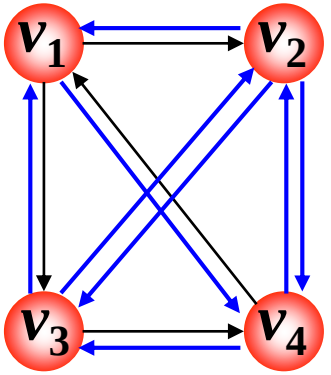
完全图：有 $n(n-1)/2$ 条边的无向图（即：每两个顶点之间都存在着一条边）称为**完全图**。



有向图中弧的取值范围：$0 \le e \le n(n-1)$。

（用 n 表示图中顶点数目，用 e 表示弧的数目。且不考虑顶点到其自身的弧。）

有向完全图：有 $n(n-1)$ 条弧的有向图（即：每两个顶点之间都存在着方向相反的两条弧）称为**有向完全图**。

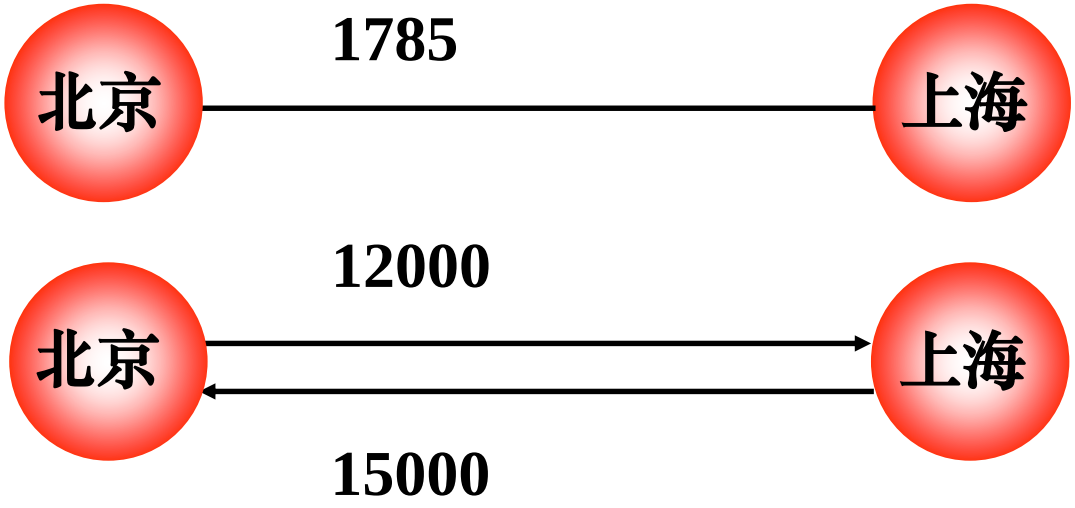


稀疏图： 含有很少条边或弧的图。

稠密图： 含有很多条边或弧的接近完全图的图。

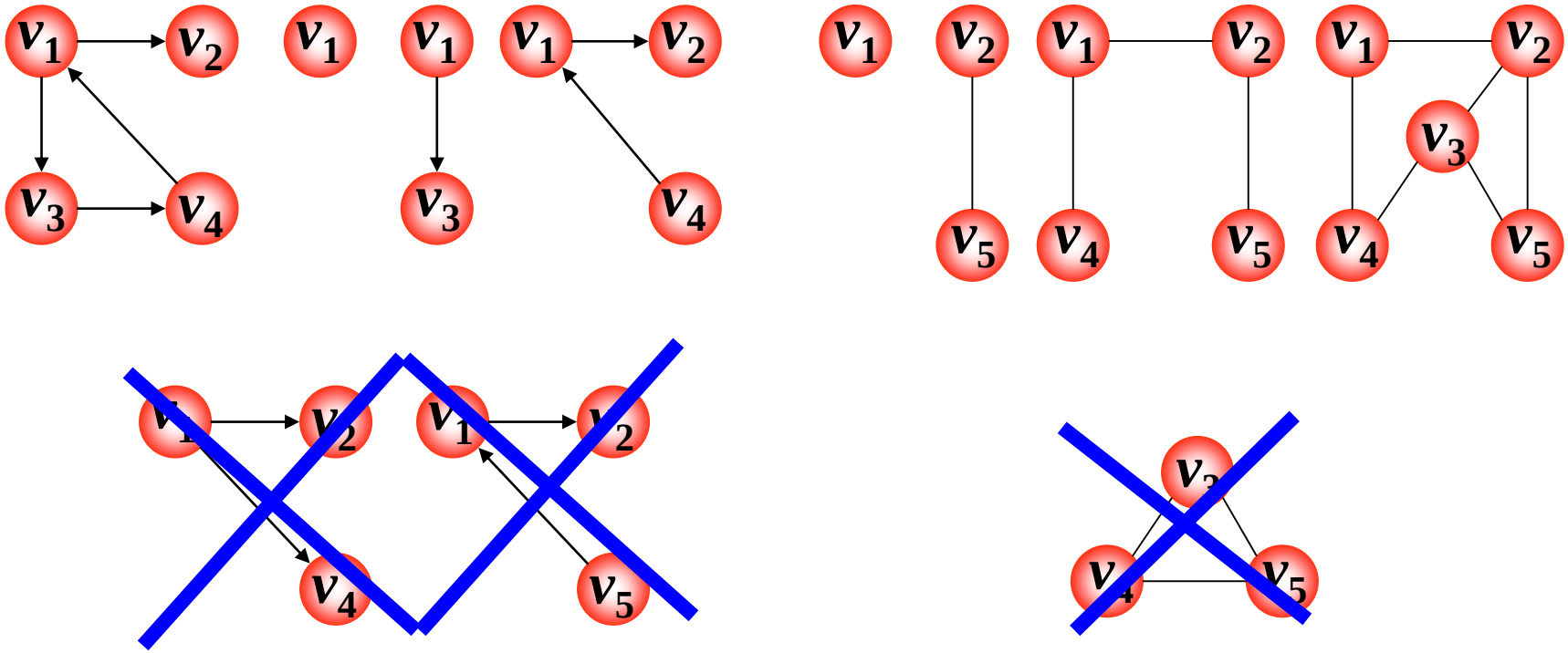
权： 与图的边或弧相关的数，这些数可以表示从一个顶点到另一个顶点的距离或耗费。

网： 带权的图。



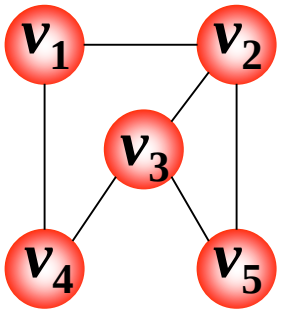
子图： 如果图 $G=(V,E)$ 和 $G'=(V',E')$，满足：

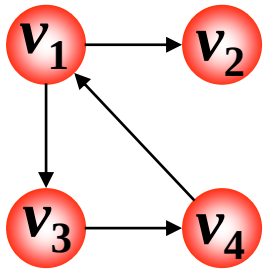
$V'\subseteq V$ 且 $E'\subseteq E$，则称 G' 为 G 的子图。



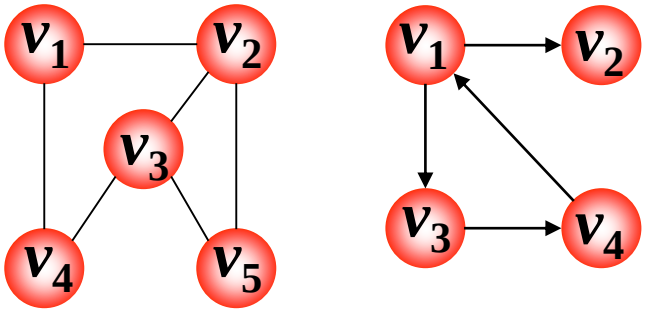
邻接点：若 (v, v') 是一条边，则称顶点 v 和 v' 互为邻接点，或称 v 和 v' 相邻接；称边 (v, v') **依附**于顶点 v 和 v'，或称 (v, v') 与顶点 v 和 v' **相关联**。

若 $<v, v'>$ 是一条弧，则称顶点 v 邻接**到** v'，顶点 v' 邻接**自**顶点 v。并称弧 $<v, v'>$ 与顶点 v 和 v' **相关联**。





度： **无向图**中顶点 v 的度是和 v 相关联的边的数目，记为：$TD(v)$。



入度： **有向图**中以顶点 v 为头的弧的数目称为 v 的入度，记为：$ID(v)$。

出度： **有向图**中以顶点 v 为尾的弧的数目称为 v 的出度，记为：$OD(v)$。

度： 入度和出度之和，即：$TD(v) = ID(v) + OD(v)$。

如果顶点 v_i 的度为 $TD(v_i)$，则一个有 n 个顶点 e 条边（弧）的图，满足如下关系：

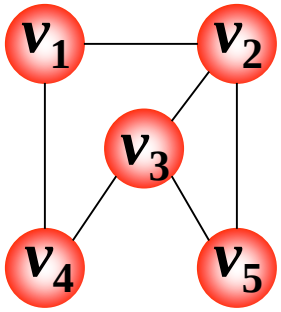
$$e = \frac{1}{2}\sum_{i=1}^{n} TD(v_i)$$

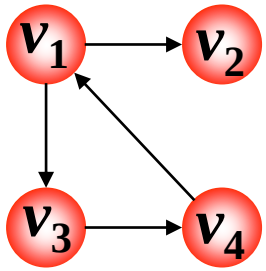
路径： 从顶点 v 到 v' 的路径是一个顶点序列

$(v = v_{i,0}, v_{i,1}, \cdots, v_{i,m} = v')$，满足 $(v_{i,j-1}, v_{i,j}) \in VR$

或 $<v_{i,j-1}, v_{i,j}> \in VR$，$(1 \le j \le m)$。

对于有向图，路径也是有向的。





路径长度： 路径上边或弧的数目。

回路（环）： 第一个顶点和最后一个顶点相同的路径。

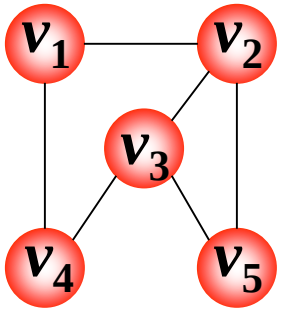
简单路径： 序列中顶点（两端点除外）不重复出现的路径。

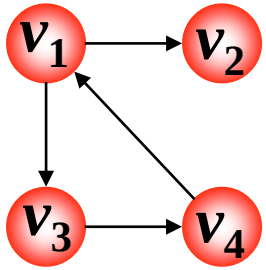
简单回路（简单环）： 前后两端点相同的简单路径。

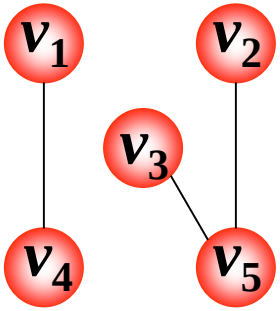
连通： 从顶点 v 到 v' 有路径，则说 v 和 v' 是连通的。

连通图： 图中任意两个顶点都是连通的。

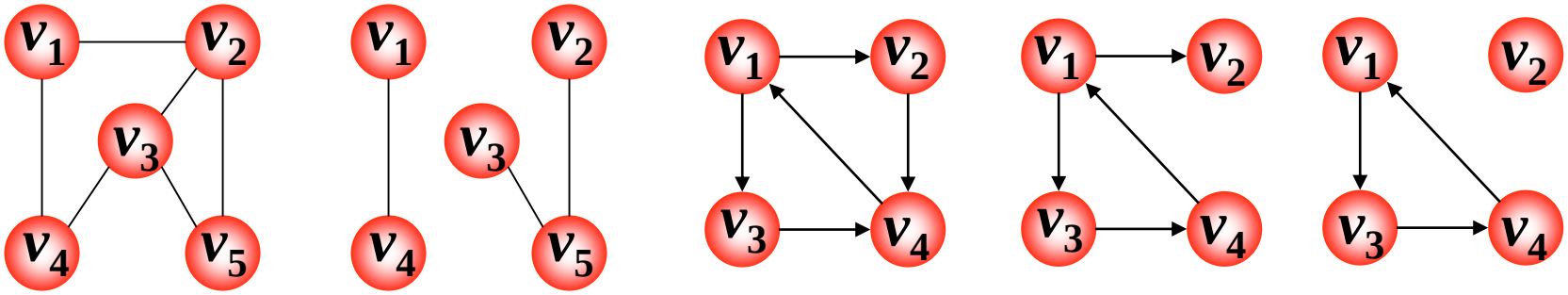
非连通图： 有 n 个顶点和小于 n-1 条边的图。







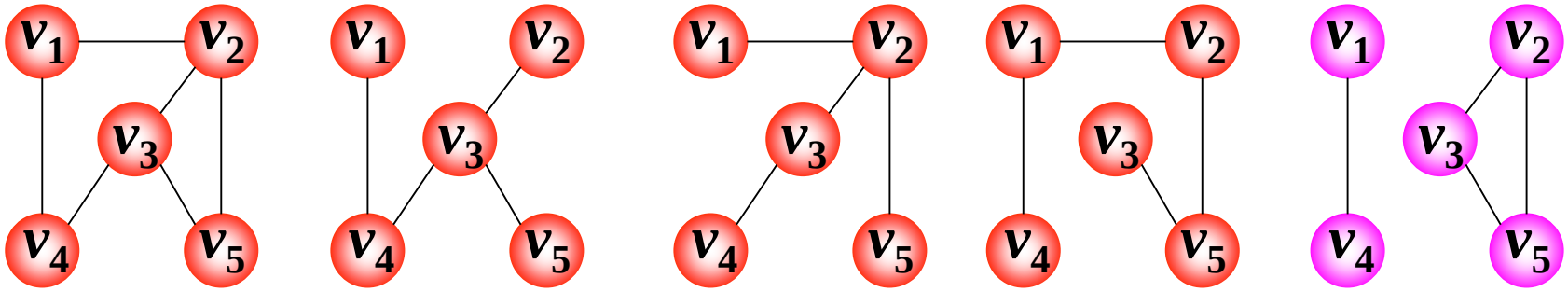
连通分量： 无向图的极大连通子图（不存在包含它的更大的连通子图）；任何连通图的连通分量只有一个，即其本身；非连通图有多个连通分量（非连通图的每一个连通部分）。



强连通图： 任意两个顶点都连通的有向图。

强连通分量： 有向图的极大强连通子图；任何强连通图的强连通分量只有一个，即其本身；非强连通图有多个强连通分量。

生成树： 所有顶点均由边连接在一起但不存在回路的图。



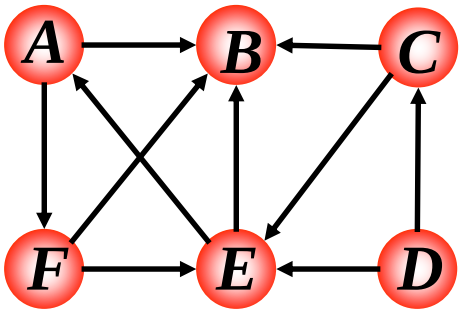
注

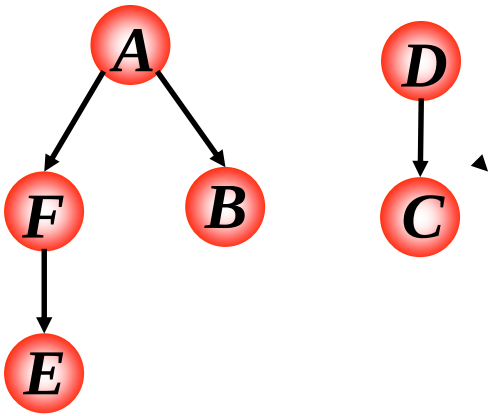
- 一个图可以有许多棵不同的生成树。
- 所有生成树具有以下共同特点：
  - 生成树的顶点个数与图的顶点个数相同；
  - 生成树是图的极小连通子图；
  - 一个有 n 个顶点的连通图的生成树有 n-1 条边；
  - 生成树中任意两个顶点间的路径是唯一的；
  - 在生成树中再加一条边必然形成回路。
- 含 n 个顶点 n-1 条边的图不一定是生成树。

生成森林：对于非连通图，其每个连通分量可以构造一棵生成树，合成起来就是一个生成森林。

有向树：如果一个有向图恰有一个顶点的入度为 0 ，其余顶点的入度均为 1 ，则是一棵有向树。

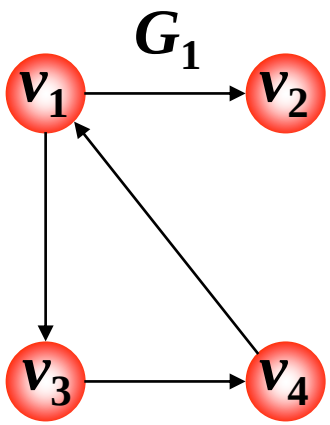
有向图的生成森林：由若干棵有向树组成，含有图中全部顶点，但只有足以构成若干棵不相交的有向树的弧。

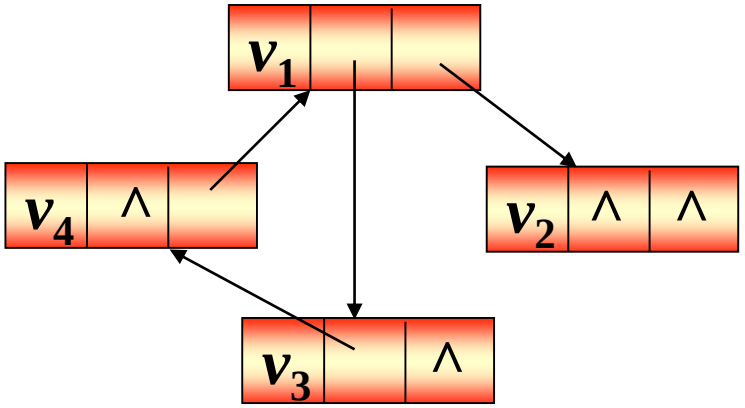


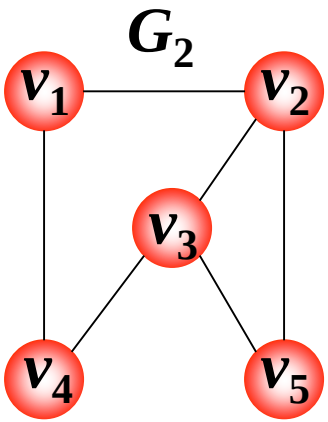


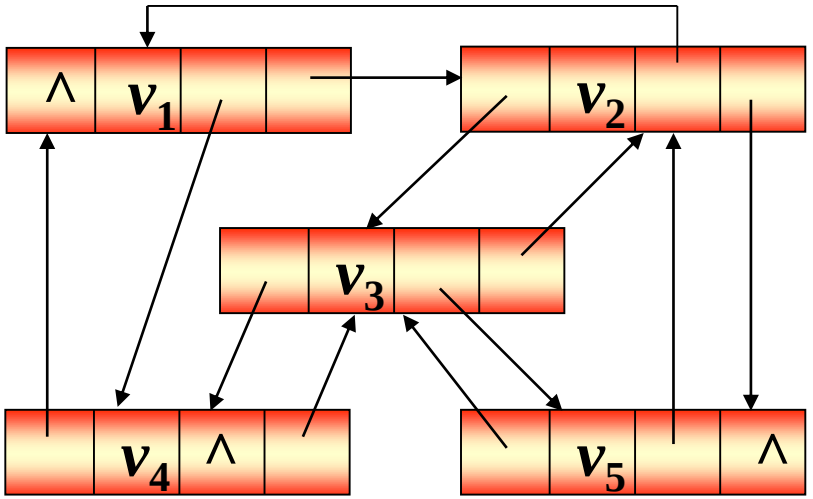
7.2 图的存储结构

- 多重链表







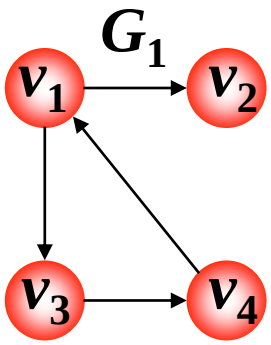


7.2.1 数组表示法（邻接矩阵表示法）

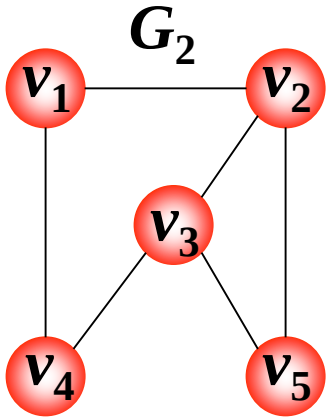
一个有 n 个顶点的图，可用两个数组存储。其中一个**一维数组**存储数据元素（**顶点**）的信息，另一个**二维数组**（**邻接矩阵**）存储数据元素之间的关系（**边或弧**）的信息。

邻接矩阵： 设 $G=(V, VR)$ 是具有 n 个顶点的图，顶点的顺序依次为 $\{v_1, v_2, \ldots, v_n\}$，则 G 的邻接矩阵是具有如下性质的 n 阶方阵：

$$A[i,j]=\begin{cases} 1: & \text{若}(v_i, v_j)\text{或}<v_i, v_j>\text{是}VR\text{中的边或弧} \\ 0: & \text{若}(v_i, v_j)\text{或}<v_i, v_j>\text{不是}VR\text{中的边或弧} \end{cases}$$



$$\begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \end{array}$$



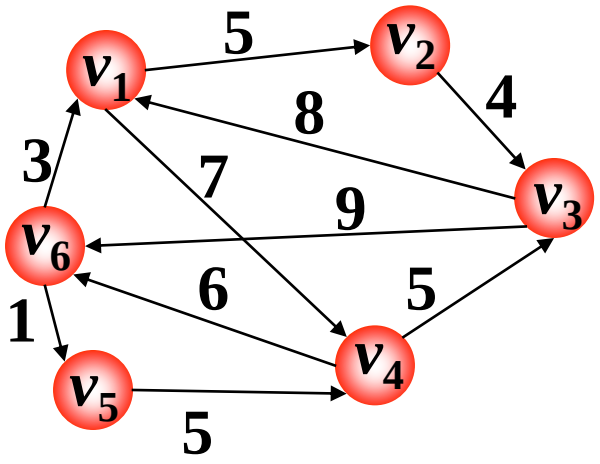
$$\begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} \begin{array}{c} \begin{array}{ccccc} v_1 & v_2 & v_3 & v_4 & v_5 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix} \end{array}$$

特点：

- 无向图的邻接矩阵对称，可压缩存储；有 n 个顶点的无向图所需存储空间为 $n(n-1)/2$。
- 有向图的邻接矩阵不一定对称；有 n 个顶点的有向图所需存储空间为 n^2，用于稀疏图时空间浪费严重。
- 无向图中顶点 v_i 的度 $TD(v_i)$ 是邻接矩阵中第 i 行 1 的个数。
- 有向图中
  - 顶点 v_i 的**出度**是邻接矩阵中第 i **行** 1 的个数。
  - 顶点 v_i 的**入度**是邻接矩阵中第 i **列** 1 的个数。

网的邻接矩阵可定义为：

$$A[i,j]=\begin{cases} w_{ij}: & \text{若}(v_i,v_j)\text{或}<v_i,v_j>\text{是}VR\text{中的边或弧} \\ \infty: & \text{若}(v_i,v_j)\text{或}<v_i,v_j>\text{不是}VR\text{中的边或弧} \end{cases}$$

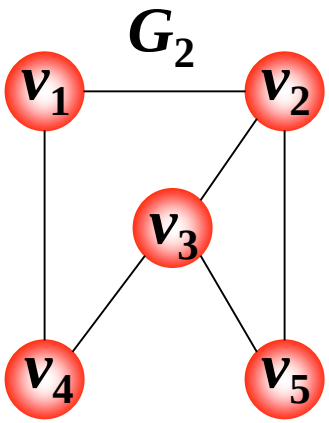


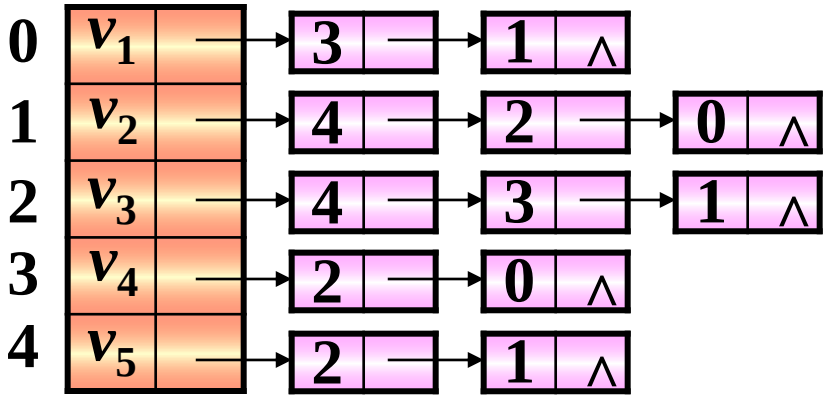
$$\begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{array}\begin{array}{c} \begin{array}{cccccc} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \end{array} \\ \begin{bmatrix} \infty & 5 & \infty & 7 & \infty & \infty \\ \infty & \infty & 4 & \infty & \infty & \infty \\ 8 & \infty & \infty & \infty & \infty & 9 \\ \infty & \infty & 5 & \infty & \infty & 6 \\ \infty & \infty & \infty & 5 & \infty & \infty \\ 3 & \infty & \infty & \infty & 1 & \infty \end{bmatrix} \end{array}$$

图的数组（邻接矩阵）存储表示：

```
#define INFINITY    INT_MAX // 最大值∞
#define MAX_VERTEX_NUM   20 // 最大顶点个数
typedef enum {DG, DN, UDG, UDN} GraphKind;
                  //{有向图,有向网,无向图,无向网}
typedef struct ArcCell {
  VRType  adj;  // VRType是顶点关系类型。对无权图用1或0表示相邻否；
                  // 对带权图，则为权值类型。
  InfoType   *info; // 该弧相关信息的指针
} ArcCell,  AdjMatrix[MAX_VERTEX_NUM][MAX_VERTEX_NUM];

typedef struct {
  VertexType   vexs[MAX_VERTEX_NUM]; // 顶点向量
  AdjMatrix     arcs; // 邻接矩阵
  int     vexnum, arcnum; // 图的当前顶点数和弧(边)数
  GraphKind   kind; // 图的种类标志
} MGraph;
```

7.2.2 邻接表（类似于树的孩子链表表示法）





头结点

data	firstarc

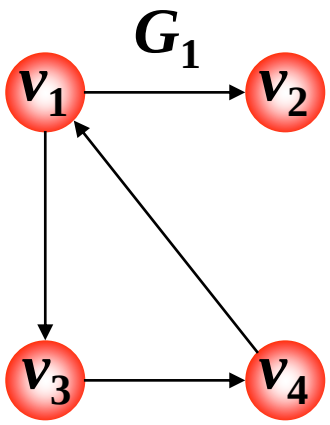
表结点

adjvex	nextarc	info

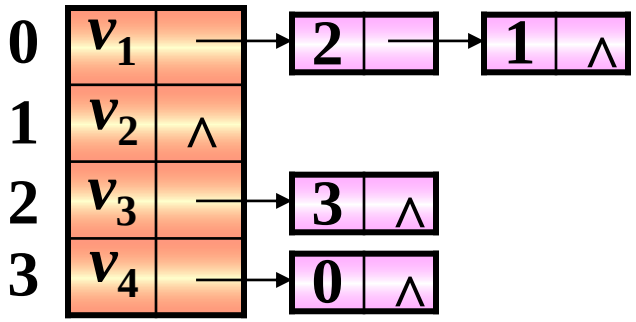
邻接点域，存放与 v_i 邻接的顶点在表头数组中的位置。

特点：

- 若无向图中有 …边或弧。…表需 n 个头结点和 $2e$ 个表结点。适宜存储稀疏图。
- 无向图中顶点 v_i 的度为第 i 个单链表中的结点数。

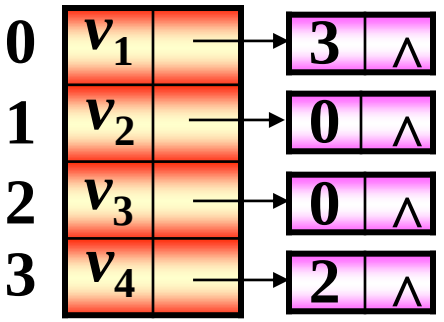


邻接表



找出度易，
找入度难。

逆邻接表



找入度易，
找出度难。

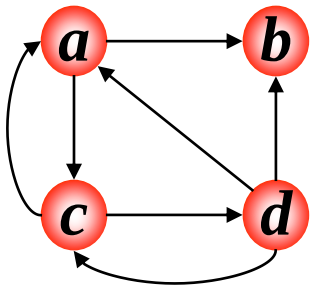
特点：

- 顶点 v_i 的**出度**为第 i 个单链表中的结点个数。
- 顶点 v_i 的**入度**为整个单链表中邻接点域值是 i-1 的结点个数。
- 顶点 v_i 的**入度**为第 i 个单链表中的结点个数。
- 顶点 v_i 的**出度**为整个单链表中邻接点域值是 i-1 的结点个数。

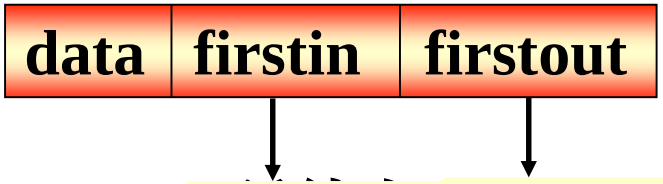
图的邻接表存储表示：

```
#define MAX_VERTEX_NUM 20
typedef struct ArcNode {
    int   adjvex; // 该弧所指向的顶点的位置
    struct ArcNode   *nextarc; // 指向下一条弧的指针
    InfoType   *info; // 该弧相关信息的指针
} ArcNode;

typedef struct VNode {
    VertexType   data; // 顶点信息
    ArcNode      *firstarc; // 指向第一条依附该顶点的弧
} VNode, AdjList[MAX_VERTEX_NUM];

typedef struct {
    AdjList   vertices;
    int   vexnum, arcnum; // 图的当前顶点数和弧数
    int   kind; // 图的种类标志
} ALGraph;
```

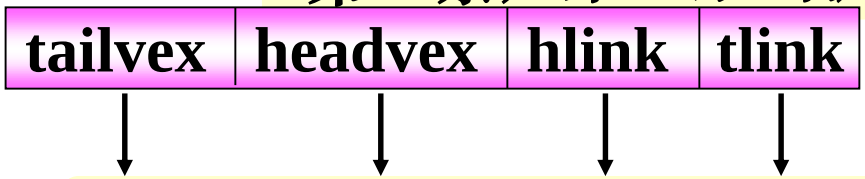
7.2.3 十字链表

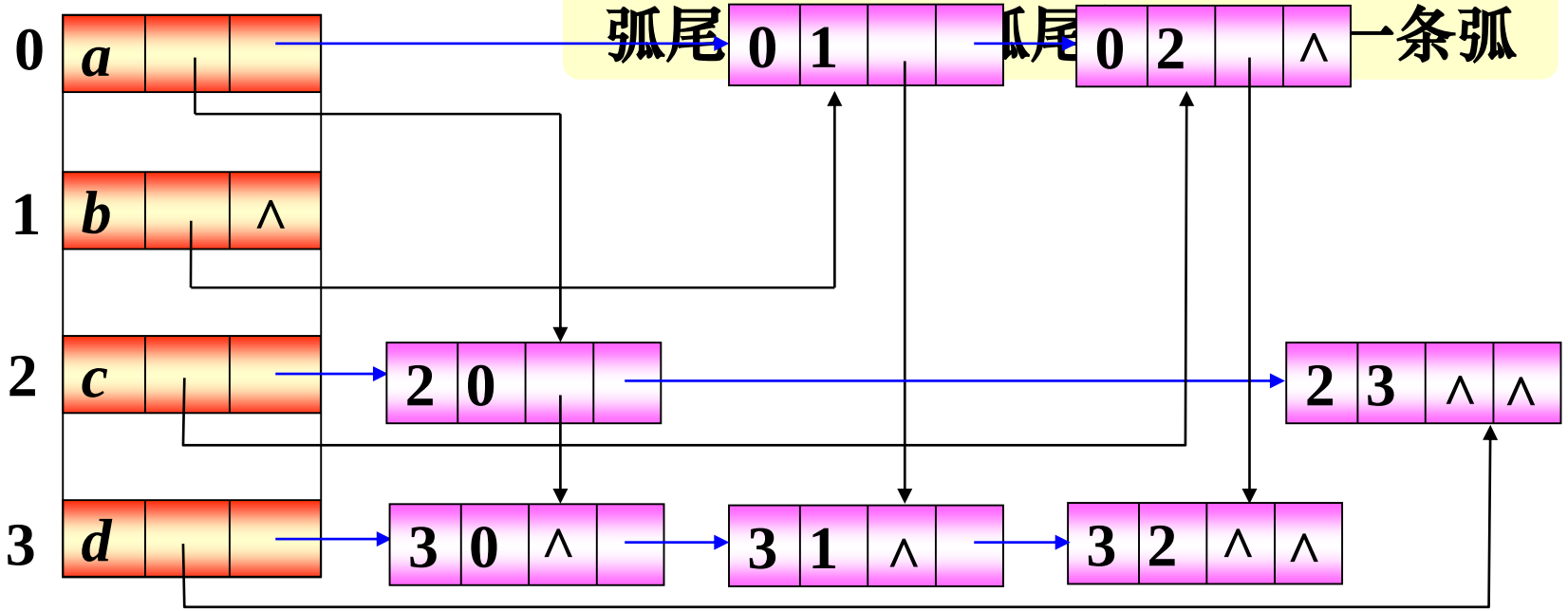


顶点结点



第一条入弧 第一条出弧





有向图的十字链表存储表示：

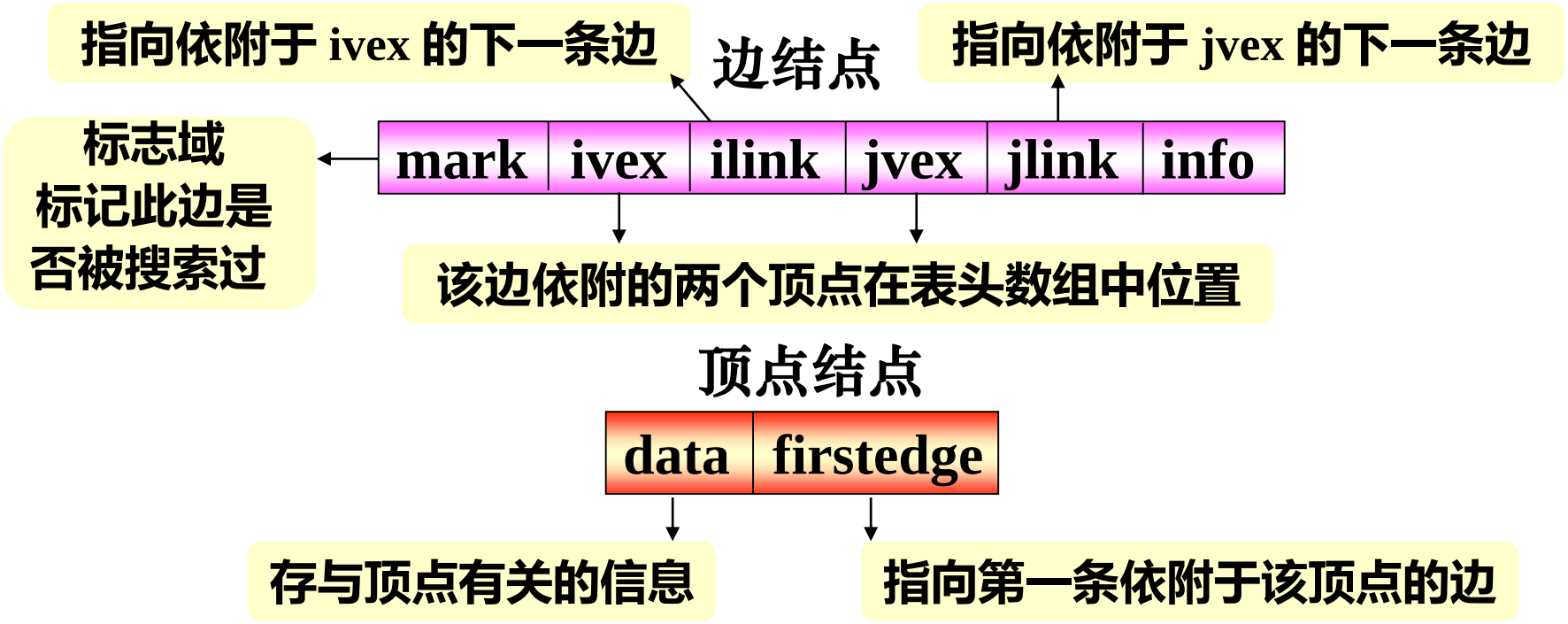
```
#define MAX_VERTEX_NUM   20
typedef struct ArcBox {
   int   tailvex, headvex; // 该弧的尾和头顶点的位置
   struct  ArcBox   *hlink,  *tlink;
                          // 分别指向下一个弧头相同和弧尾相同的弧的指针域
   InfoType *info;    // 该弧相关信息的指针
} ArcBox;

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;
```

7.2.3 邻接多重表（无向图的另一种链式存储结构）

邻接表优点：容易求得顶点和边的信息。

缺点：某些操作不方便（如：删除一条边需找表示此边的两个结点）。

邻接多重表：每条边用一个结点表示。其结点结构如下：



指向依附于 ivex 的下一条边

边结点

指向依附于 jvex 的下一条边

标志域
标记此边是
否被搜索过

mark	ivex	ilink	jvex	jlink	info

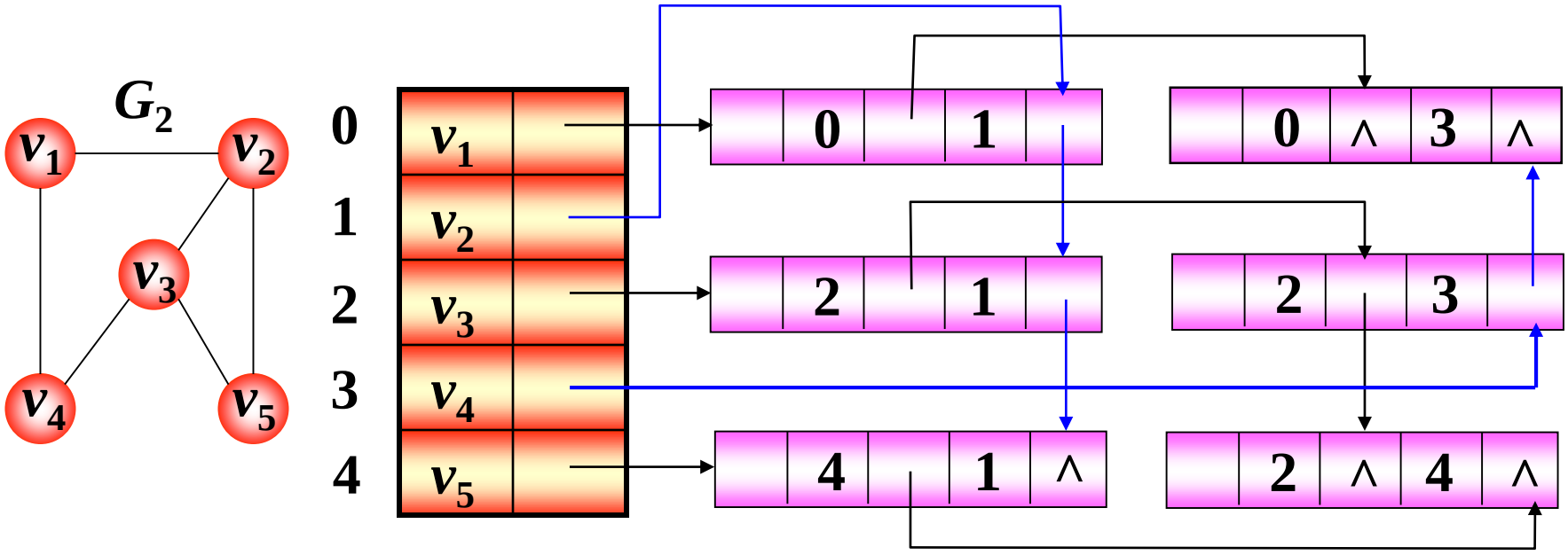
该边依附的两个顶点在表头数组中位置

顶点结点

存与顶点有关的信息

data	firstedge

指向第一条依附于该顶点的边

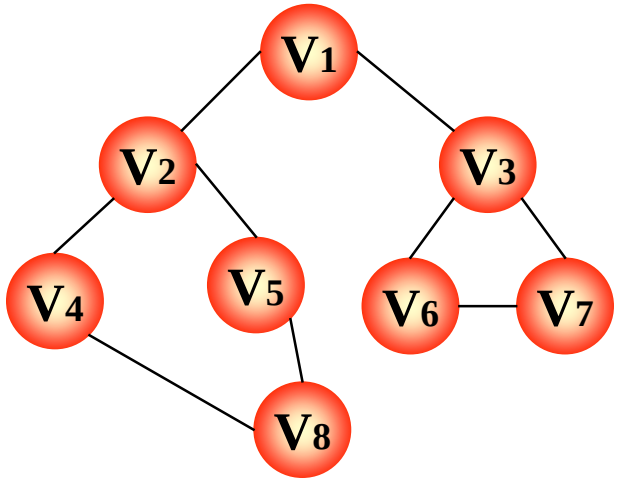


无向图的邻接多重表存储表示：

```
#define MAX_VERTEX_NUM 20
typedef emnu {unvisited, visited} VisitIf;
typedef struct Ebox {
  VisitIf  mark; // 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;
```


7.3 图的遍历

从图的任意指定顶点出发，依照某种规则去访问图中所有顶点，且每个顶点仅被访问一次，这一过程叫做**图的遍历**。



图的遍历按照深度优先和广度优先规则去实施，通常有**深度优先遍历法**（Depth_First Search——DFS ）和 **广度优先遍历法**（ Breadth_Frist Search——BFS）两种。

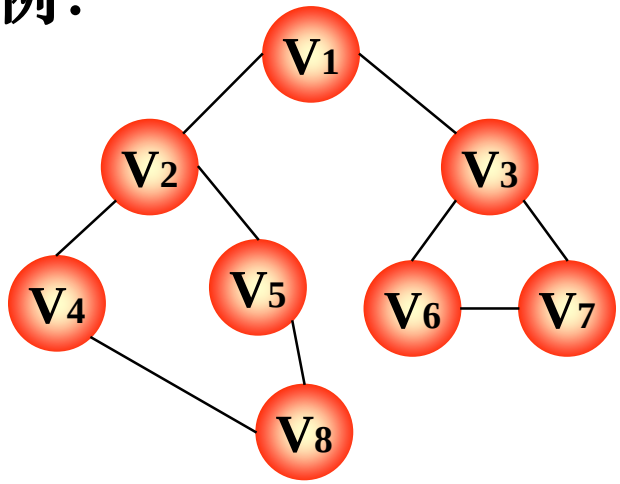
7.3.1 深度优先遍历（DFS）

方法： 1、访问指定的起始顶点；

2、若当前访问的顶点的邻接顶点有未被访问的，则任选一个访问之；反之，退回到最近访问过的顶点；直到与起始顶点相通的全部顶点都访问完毕；

3、若此时图中尚有顶点未被访问，则再选其中一个顶点作为起始顶点并访问之，转2；反之，遍历结束。

例：



顶点访问次序：

V1⇒V2⇒V4⇒V8⇒V5⇒V3⇒V6⇒V7

V1⇒V2⇒V5⇒V8⇒V4⇒V3⇒V6⇒V7

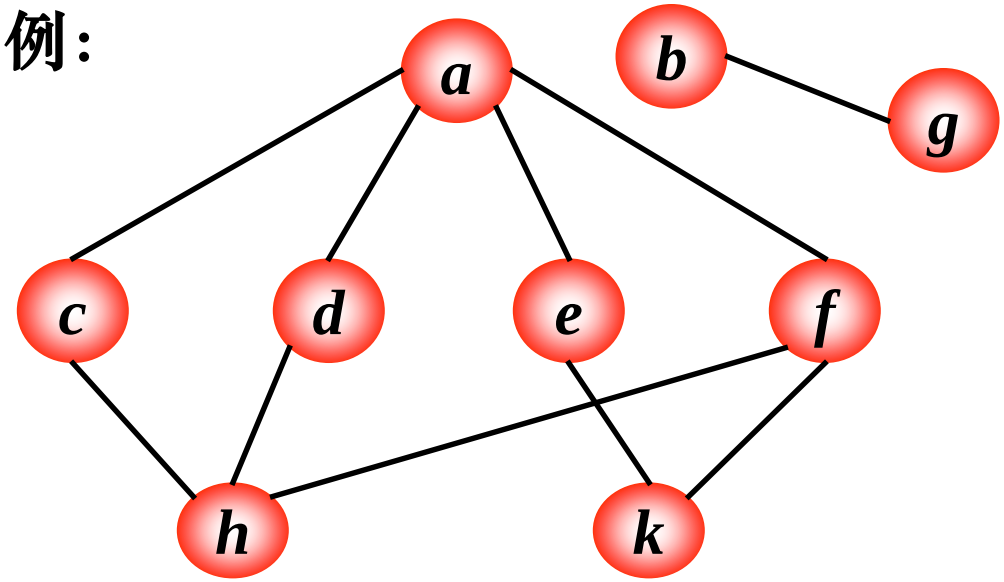
V1⇒V2⇒V4⇒V8⇒V5⇒V3⇒V7⇒V6

V1⇒V2⇒V5⇒V8⇒V4⇒V3⇒V7⇒V6

V1⇒V3⇒V6⇒V7⇒V2⇒V4⇒V8⇒V5

连通图的深度优先遍历类似于树的先根遍历

例：



顶点访问次序：

a c h d f k e b g

a c h d f k e g b

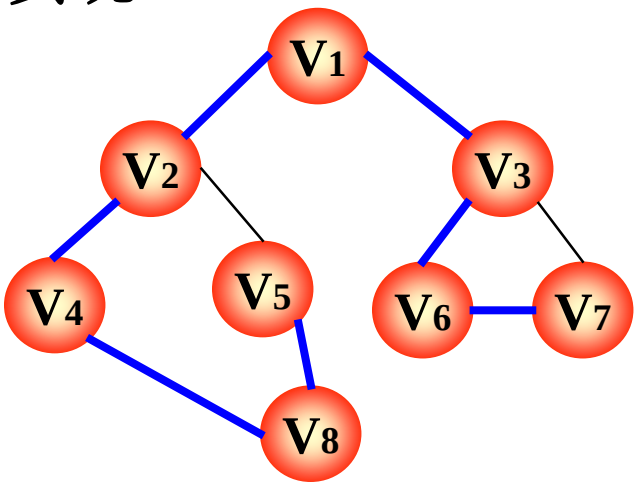
a c h f k e d b g

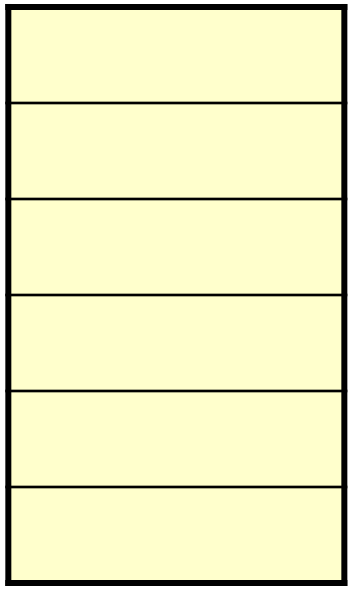
如何判别V的邻接点是否被访问？

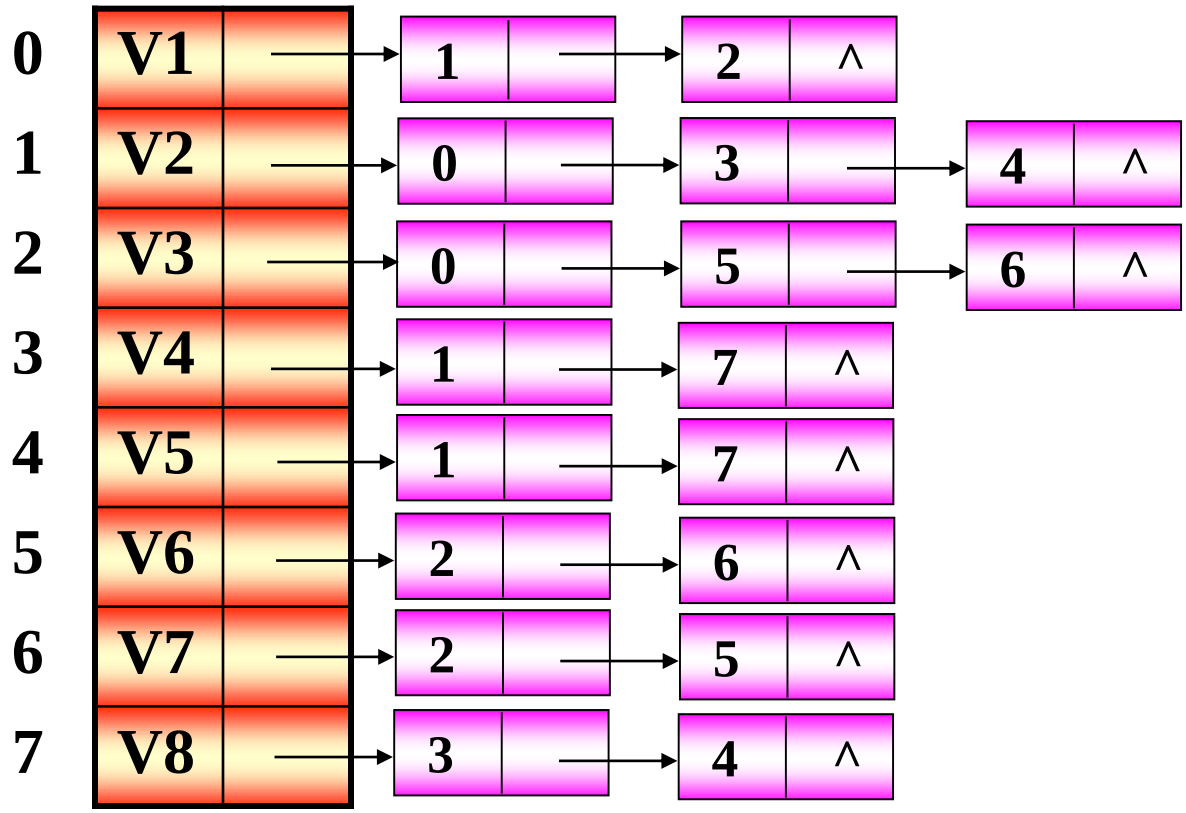
解决办法：为每个顶点设立一个“访问标志”。

首先将图中每个顶点的访问标志设为 FALSE，之后搜索图中每个顶点，如果未被访问，则以该顶点为起始点，进行深度优先遍历，否则继续检查下一顶点。

实现：

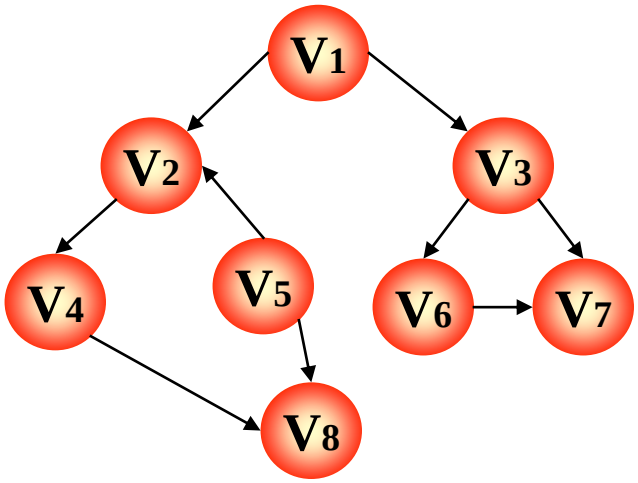


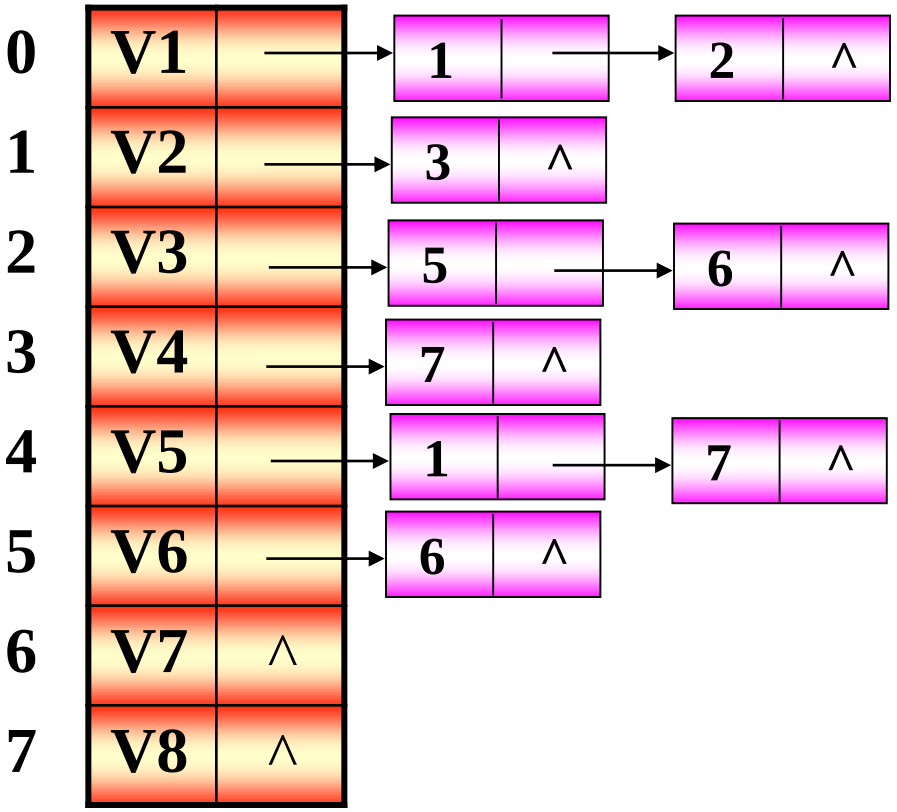




0	1	2	3	4	5	6	7
1	1	1	1	1	1	1	1

V1⇒V2⇒V4⇒V8⇒V5⇒V3⇒V6⇒V7





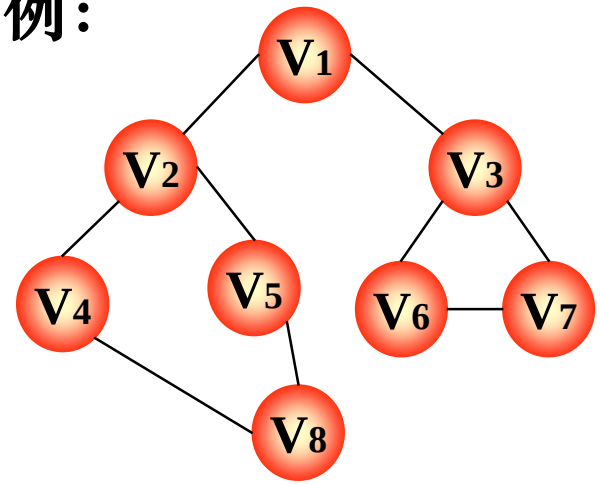
0	1	2	3	4	5	6	7
1	1	1	1	1	1	1	1

V1⇒V2⇒V4⇒V8⇒V3⇒V6⇒V7⇒V5

7.3.2 广度优先遍历（BFS）

方法：从图的某一结点出发，首先依次访问该结点的所有邻接顶点 $Vi_1, Vi_2, \ldots, Vi_n$ 再按这些顶点被访问的先后次序依次访问与它们相邻接的所有未被访问的顶点，重复此过程，直至所有顶点均被访问为止。

例：



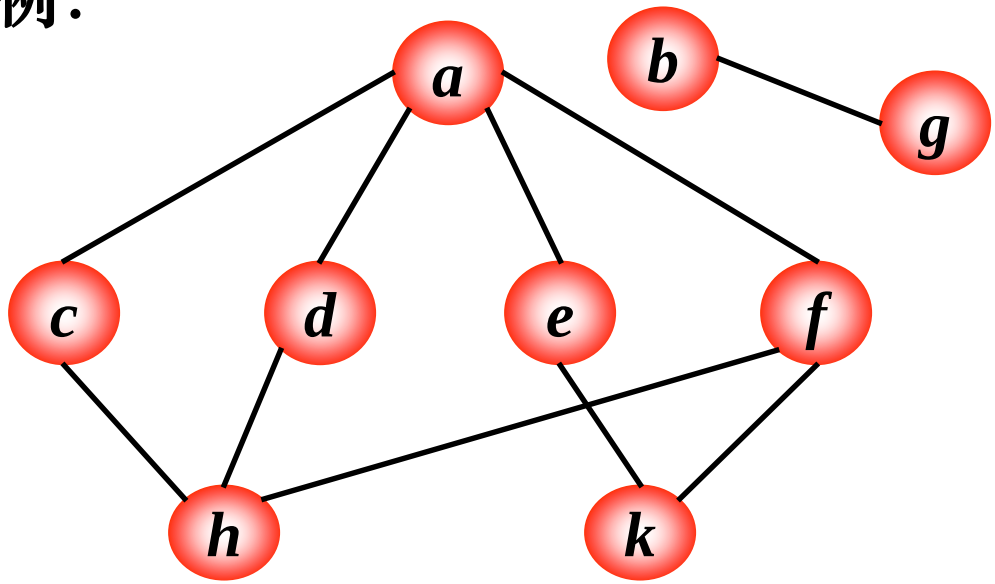
广度优先遍历：

V1⇒V2⇒V3⇒V4⇒V5⇒V6⇒V7⇒V8

V1⇒V3⇒V2⇒V7⇒V6⇒V5⇒V4⇒V8

V1⇒V2⇒V3⇒V5⇒V4⇒V7⇒V6⇒V8

例：

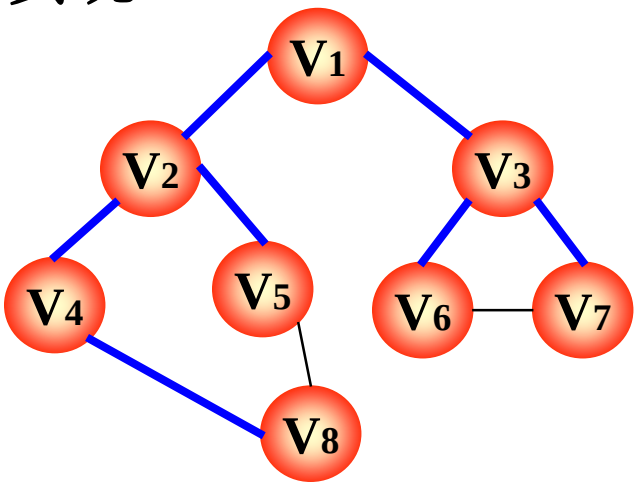


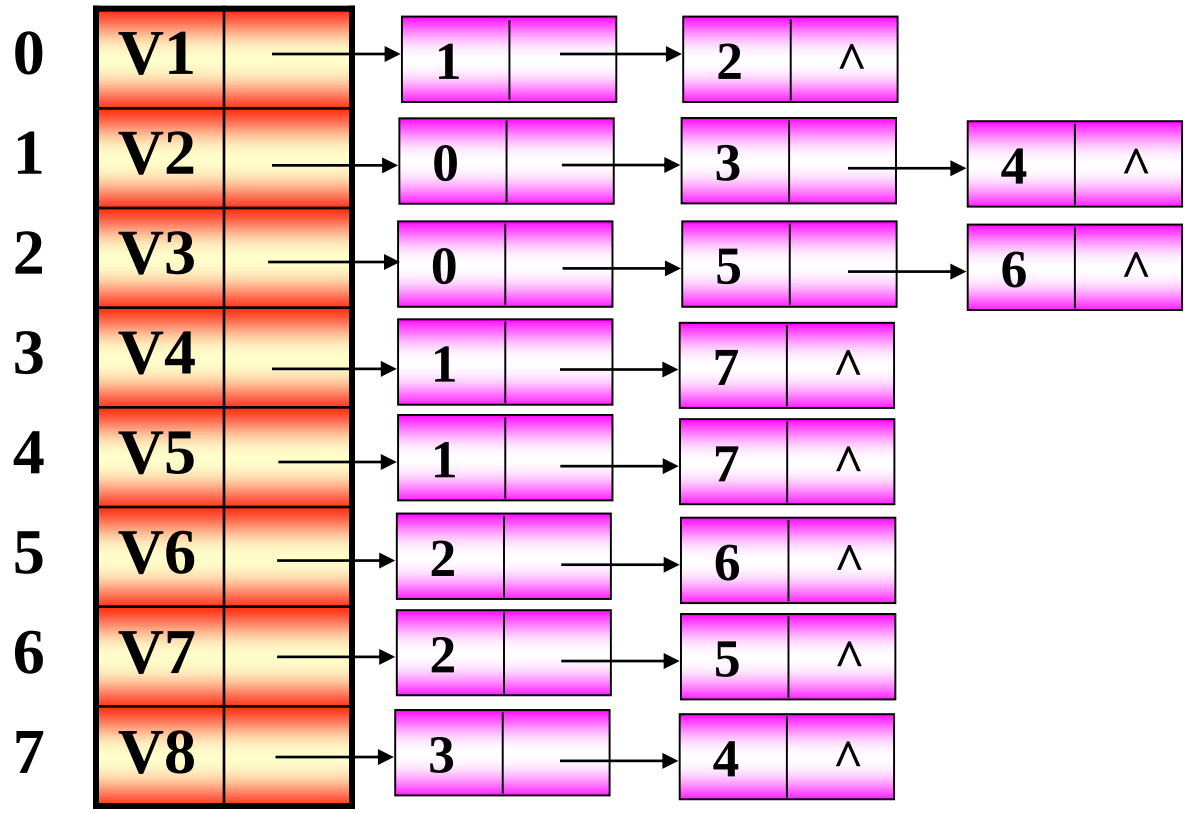
顶点访问次序：

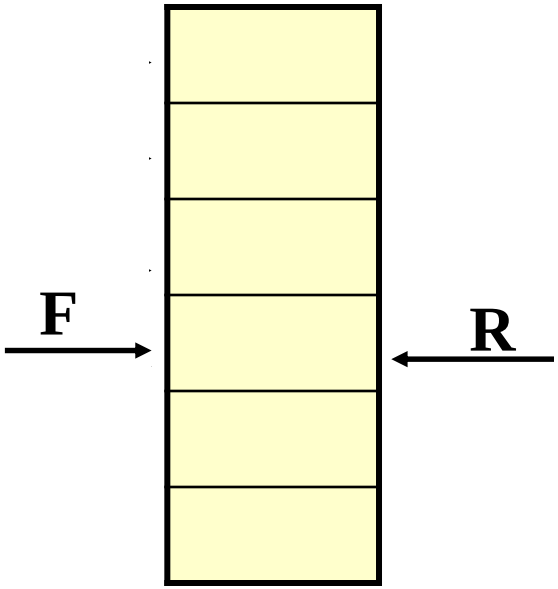
a c d e f h k b g

a c d e f h k g b

实现：



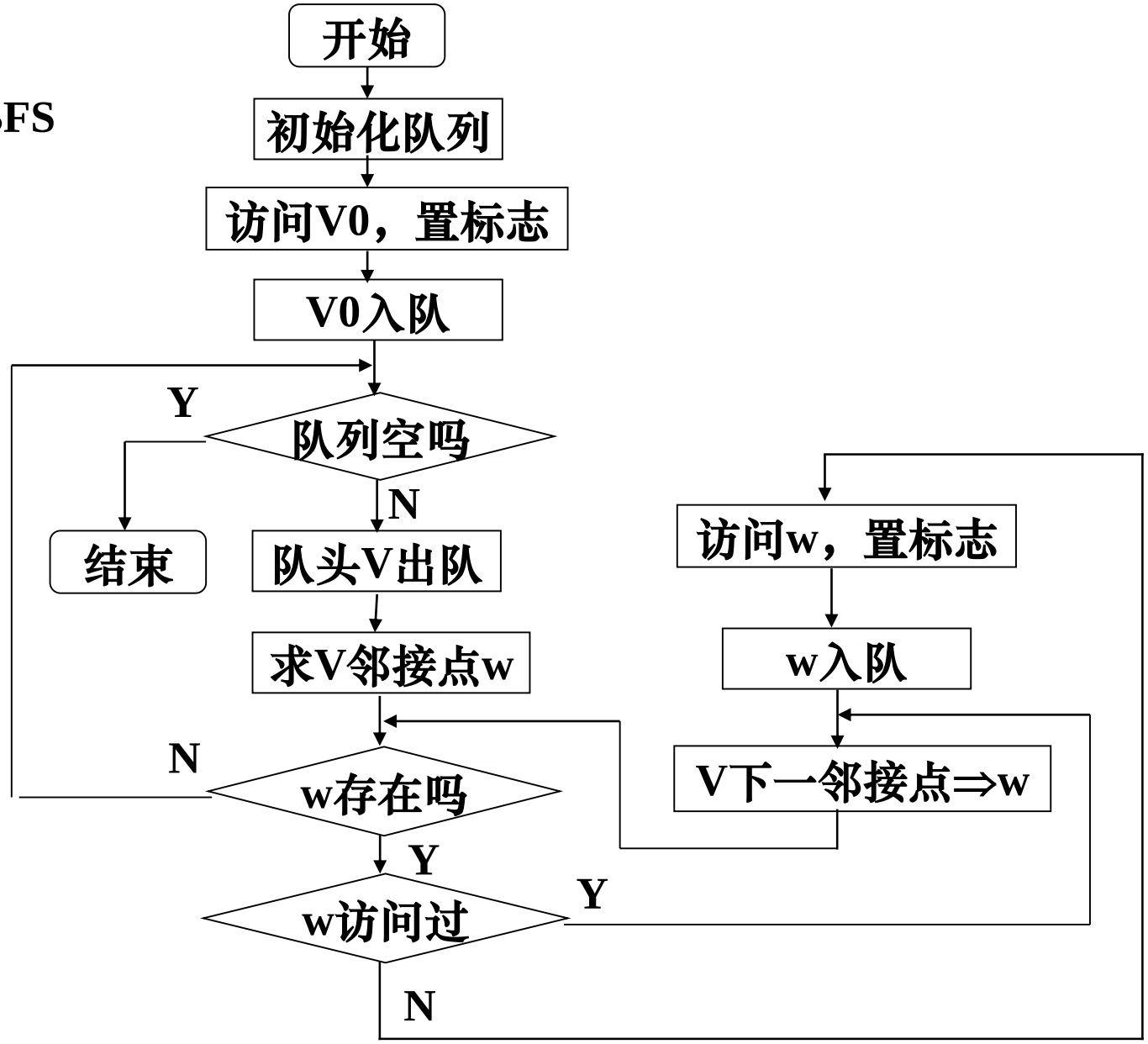




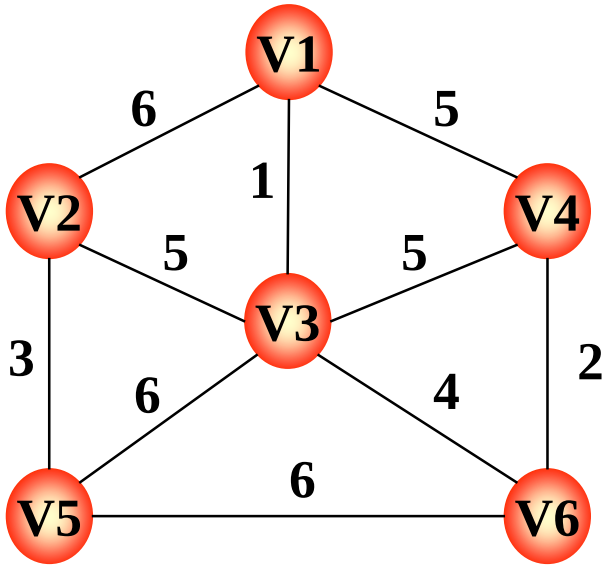
0	1	2	3	4	5	6	7
1	1	1	1	1	1	1	1

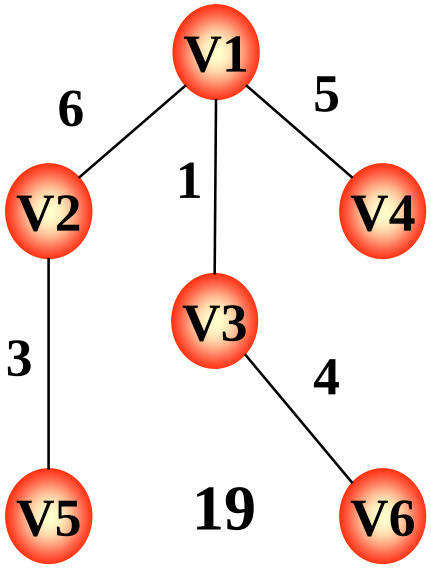
V1⇒V2⇒V3⇒V4⇒V5⇒V6⇒V7⇒V8

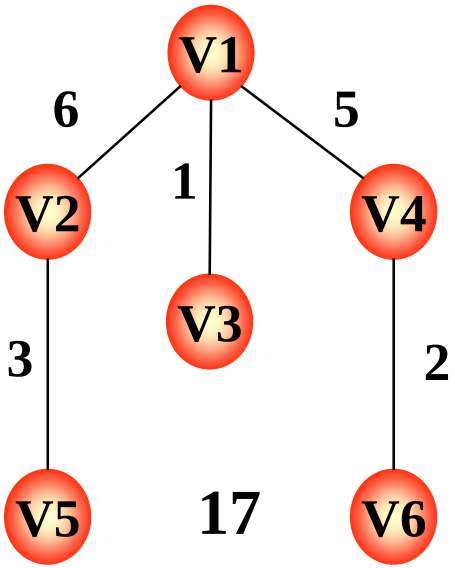
BFS


开始
初始化队列
访问V0，置标志
V0入队
Y
队列空吗
N
结束
队头V出队
求V邻接点w
N
w存在吗
Y
w访问过
Y
N
访问w，置标志
w入队
V下一邻接点⇒w

7.4.3 最小生成树







最小生成树：给定一个无向网络，在该网的所有生成树中，使得各边权数之和最小的那棵生成树称为该网的最小生成树，也叫**最小代价生成树**。

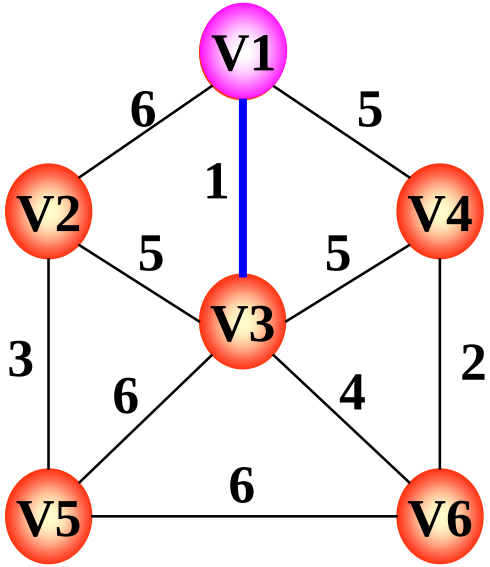
问题提出： 要在 n 个城市间建立通信联络网。**顶点：** 表示城市，**权：** 城市间通信线路的花费代价。希望此通信网花费代价最小。

问题分析： 答案只能从生成树中找，因为要做到任何两个城市之间有线路可达，**通信网必须是连通的**；但对长度最小的要求可以知道**网中显然不能有圈**，如果有圈，去掉一条边后，并不破坏连通性，但总代价显然减少了，这与总代价最小的假设是矛盾的。

结论： 希望找到一棵生成树，它的每条边上的权值之和（即建立该通信网所需花费的总代价）最小 —— 最小代价生成树。

构造最小生成树的算法很多，其中多数算法都利用了一种称之为 MST 的性质。

MST 性质： 设 N = (V, E) 是一个连通网，U 是顶点集 V 的一个非空子集。若边 (u, v) 是一条具有最小权值的边，其中u∈U，v∈V-U，则必存在一棵包含边 (u, v) 的最小生成树。



N = (V, {E})

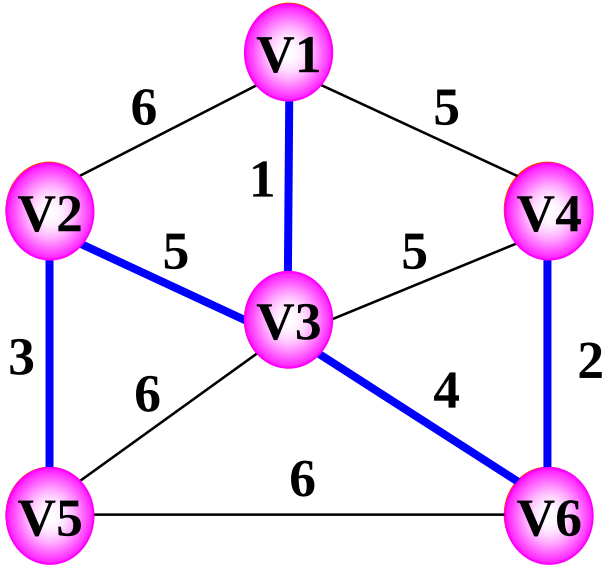
V = {v1, v2, v3, v4, v5, v6}

E = {(v1, v2), (v1, v3), (v1, v4),
(v2, v3), (v2, v5), (v3, v4),
(v3, v5), (v3, v6), (v4, v6),
(v5, v6)}

U = {v1}

构造最小生成树方法：

方法一：普里姆 (Prim) 算法。



算法思想：

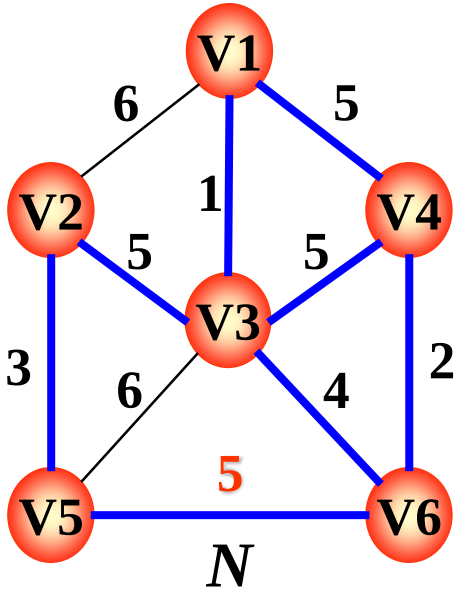
- 设 $N=(V, E)$ 是连通网，TE 是 N 上最小生成树中边的集合。
- 初始令 $U=\{u_0\}$, $(u_0 \in V)$, $TE=\{\}$。
- 在所有 $u \in U, v \in V-U$ 的边 $(u, v) \in E$ 中，找一条代价最小的边 (u_0, v_0)。
- 将 (u_0, v_0) 并入集合 TE，同时 v_0 并入 U。
- 重复上述操作直至 $U=V$ 为止，则 $T=(V, TE)$ 为 N 的最小生成树。

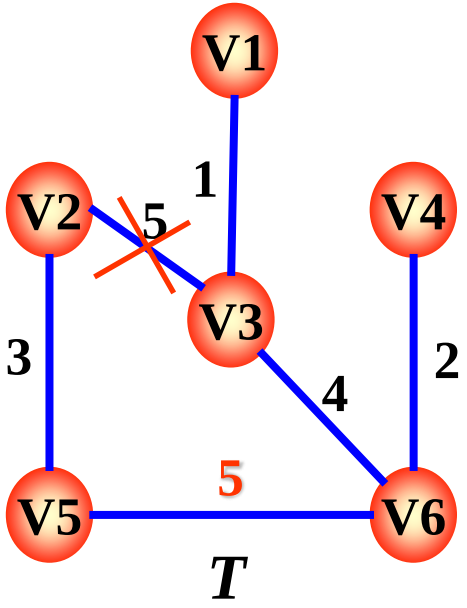
方法二：克鲁斯卡尔 (Kruskal) 算法。

算法思想：

- 设连通网 $N=(V,E)$，令最小生成树初始状态为只有 n 个顶点而无边的非连通图 $T=(V,\{\})$，每个顶点自成一个连通分量。
- 在 E 中选择代价最小的边，若该边依附的顶点落在 T 中不同的连通分量上（即：不能形成环路），则将此边加到 T 中；否则，舍去此边，选取下一条代价最小的边。
- 依此类推，直至 T 中所有顶点都在同一连通分量上为止。

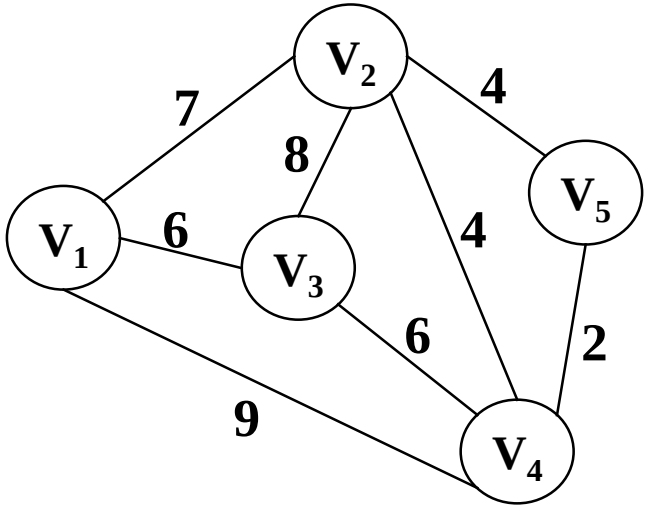
最小生成树可能不惟一





课堂练习

1. n 个顶点的连通图至少 ( ) 条边。
2. 在一个无向图的邻接表中，若表结点的个数是 m，则图中边的条数是 ( ) 条。
3. 分别用普里姆和克鲁斯卡尔算法构造下图所示网络的最小生成树。



4. 分别求出图 4 从 v2 出发按深度优先搜索和广度优先搜索算法遍历得到的顶点序列（假设图的存储结构采用邻接矩阵表示）。

5. 已知一个有向图的邻接表如图 5 所示，求出根据深度优先搜索算法从顶点 v1 出发遍历得到的顶点序列。

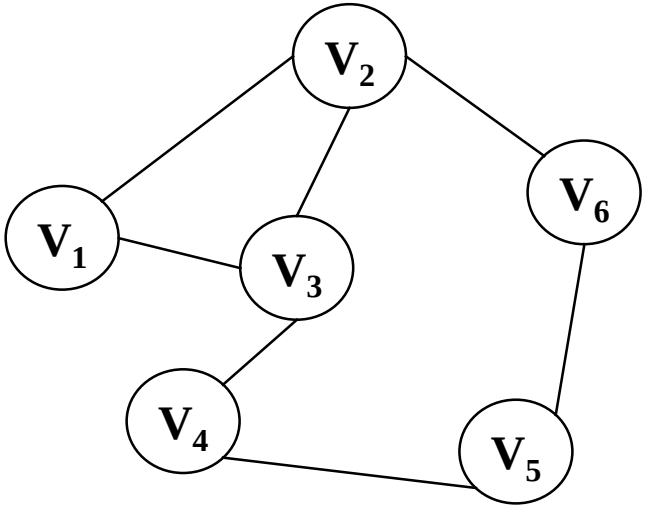


图 4

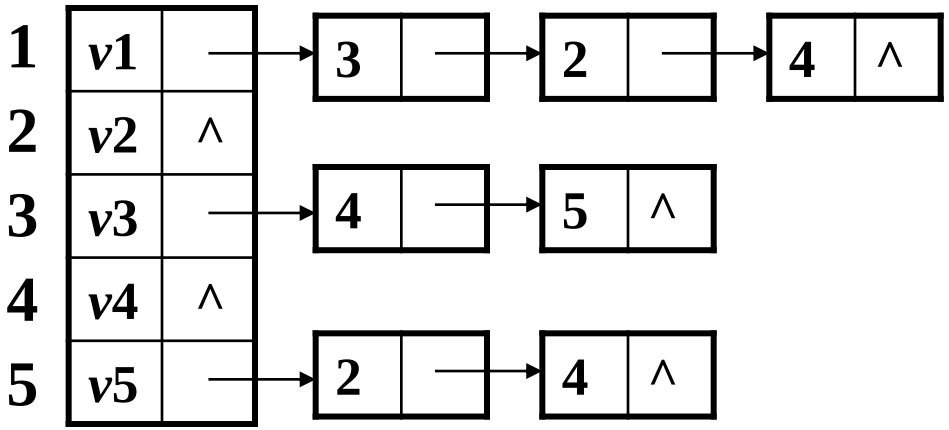


图 5

选择题

1. 在一个图中，所有顶点的度数之和等于所有边数的 ( ) 倍。

（A）1/2 （B）1 （C）2 （D）4

2. 在一个有向图中，所有顶点的入度之和等于所有顶点的出度之和的 ( ) 倍。

（A）1/2 （B）1 （C）2 （D）4

3. 一个有 n 个顶点的无向图最多有 ( ) 条边。

（A）n （B）$n(n-1)$ （C）$n(n-1)/2$ （D）$2n$

4. 具有 4 个顶点的无向完全图有 ( ) 条边。

（A）6 （B）12 （C）16 （D）20

5. 具有 6 个顶点的无向图至少应有 ( ) 条边才能确保是一个连通图。

（A）5 （B）6 （C）7 （D）8

6. 在一个具有 n 个顶点的无向图中，要连通全部顶点至少需要 ( ) 条边。

（A）n （B）$n+1$ （C）$n-1$ （D）$n/2$

7. 对于一个具有 n 个顶点的无向图，若采用邻接矩阵表示，则该矩阵的大小为 ( )。

（A）n （B）$(n-1)2$ （C）$n-1$ （D）n^2

8. 对于一个具有 n 个顶点和 e 条边的无向图，若采用邻接表表示，则表头数组的大小为 ( )，所有邻接表中表结点的总数是 ( )。

①（A）n （B）$n+1$ （C）$n-1$ （D）$n+e$

②（A）$e/2$ （B）e （C）$2e$ （D）$n+e$

9. 图的深度优先遍历算法类似于树的（ ）。

（A）先根遍历 （B）后根遍历 （C）按层遍历

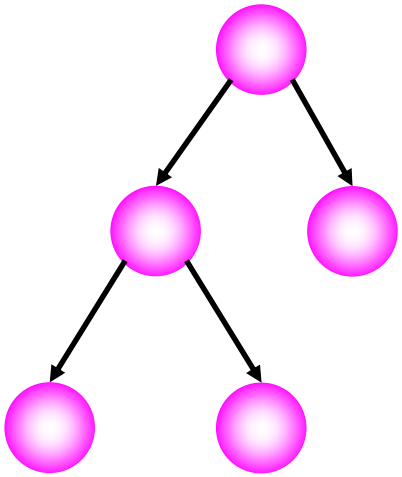
10. 图的广度优先遍历算法类似于树的（ ）。

（A）先根遍历 （B）后根遍历 （C）按层遍历

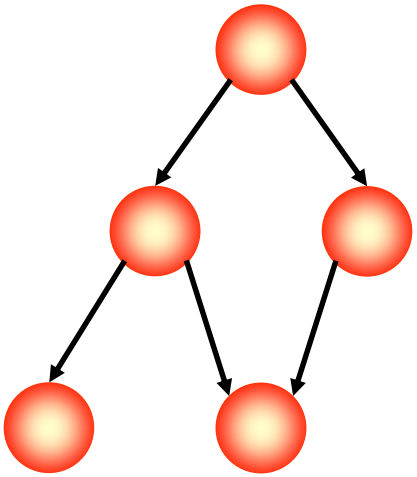
7.5 有向无环图及其应用

有向无环图：**无环的有向图，**

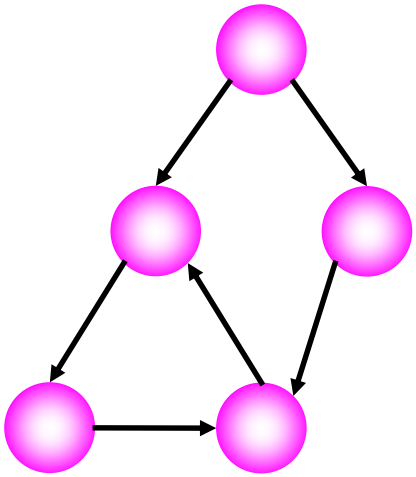
简称 DAG (Directed Acycline Graph) 图。



有向树



有向无环图

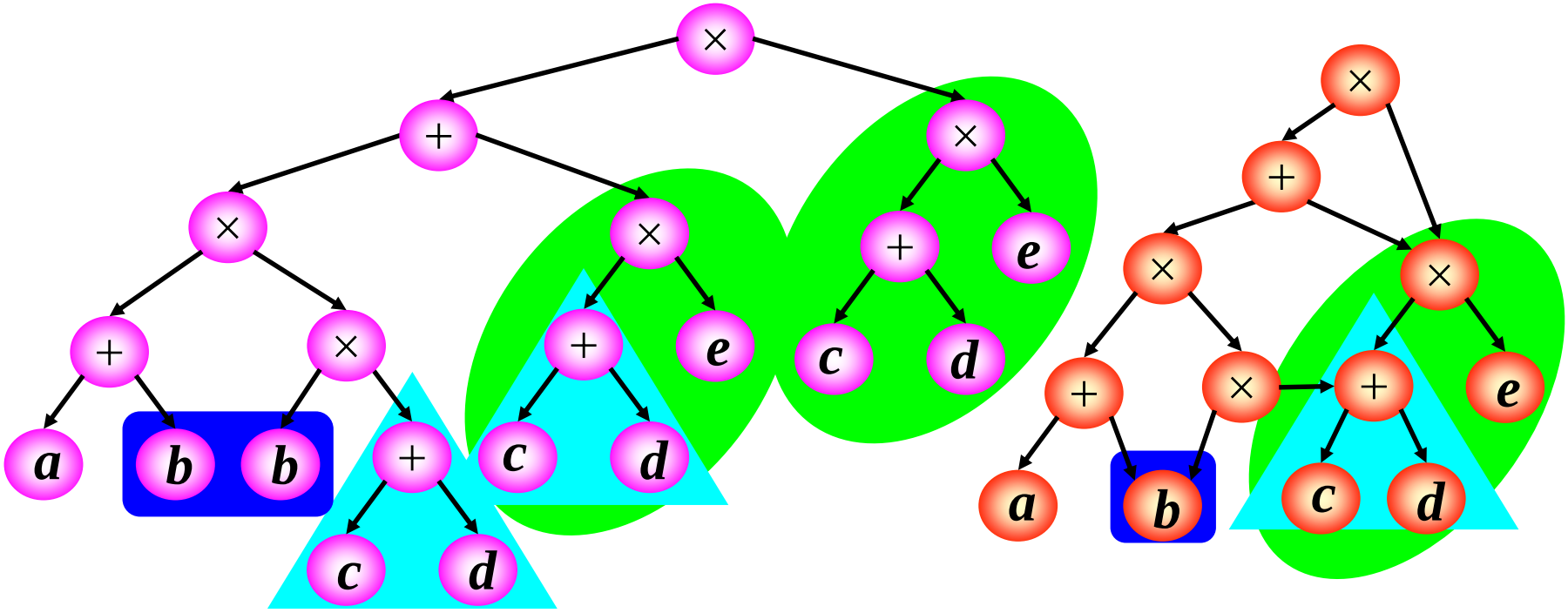


有向图

有向无环图的用途：

1、描述含有公共子式的表达式：

$$\{(a+b)\times[b\times(c+d)]+(c+d)\times e\}\times[(c+d)\times e]$$



共享相同子式　节省存储空间

2、在工程计划和管理方面的应用

除最简单的情况之外，几乎所有的工程都可分为若干个称作“活动”的子工程，并且这些子工程之间通常受着一定条件的约束，例如：其中某些子工程必须在另一些子工程完成之后才能开始。

对整个工程和系统，人们关心的是两方面的问题：

一是工程能否顺利进行；

二是完成整个工程所必须的最短时间。

对应到有向图即为进行拓扑排序和求关键路径。

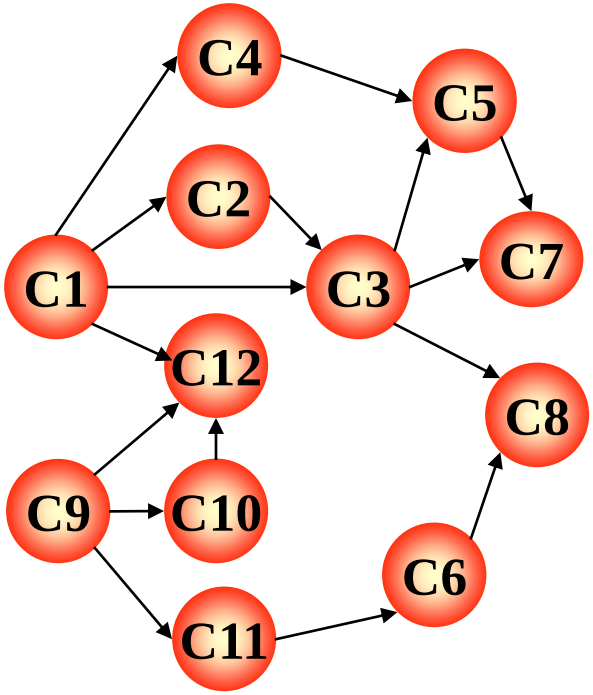
7.5.1 拓扑排序

AOV 网：

用一个有向图表示一个工程的各子工程及其相互制约的关系，其中以顶点表示活动，弧表示活动之间的优先制约关系，称这种有向图为顶点表示活动的网，简称AOV (Activity On Vertex network)网。

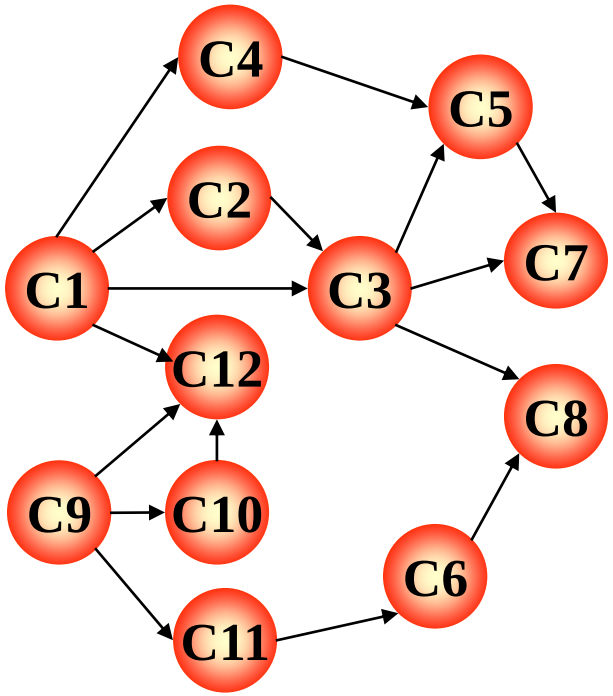
例：排课表。

课程代号	课程名称	先修课
C1	程序设计基础	无
C2	离散数学	C1
C3	数据结构	C1,C2
C4	汇编语言	C1
C5	语言的设计和分析	C3,C4
C6	计算机原理	C11
C7	编译原理	C3,C5
C8	操作系统	C3,C6
C9	高等数学	无
C10	线性代数	C9
C11	普通物理	C9
C12	数值分析	C1,C9,C10



AOV 网的特点：

- 若从 i 到 j 有一条有向路径，则 i 是 j 的前驱；j 是 i 的后继。
- 若 $<i,j>$ 是网中有向边，则 i 是 j 的直接前驱；j 是 i 的直接后继。
- **AOV 网中不允许有回路**，因为如果有回路存在，则表明某项活动以自己为先决条件，显然这是荒谬的。



问题： 如何判别 AOV 网中是否存在回路？

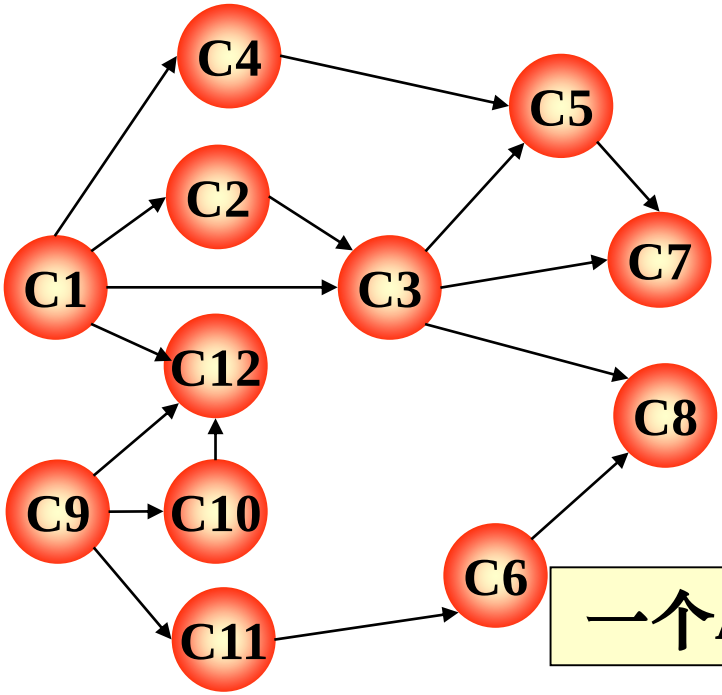
拓扑排序：

在 AOV 网没有回路的前提下，我们将全部活动排列成一个线性序列，使得若 AOV 网中有弧 <i, j> 存在，则在这个序列中，i 一定排在 j 的前面，具有这种性质的线性序列称为拓扑有序序列，相应的拓扑有序排序的算法称为拓扑排序。

检测 AOV 网中是否存在环方法： 对有向图构造其顶点的拓扑有序序列，若网中所有顶点都在它的拓扑有序序列中，则该 AOV 网必定不存在环。

拓扑排序的方法：

- 在有向图中选一个没有前驱的顶点且输出之。
- 从图中删除该顶点和所有以它为尾的弧。
- 重复上述两步，直至全部顶点均已输出；或者当图中不存在无前驱的顶点为止。



拓扑序列：

C1, C2, C3, C4, C5, C7, C9,

C10, C11, C6, C12, C8

C9, C10, C11, C6, C1, C12,

C4, C2, C3, C5, C7, C8

一个AOV网的拓扑序列不是唯一的

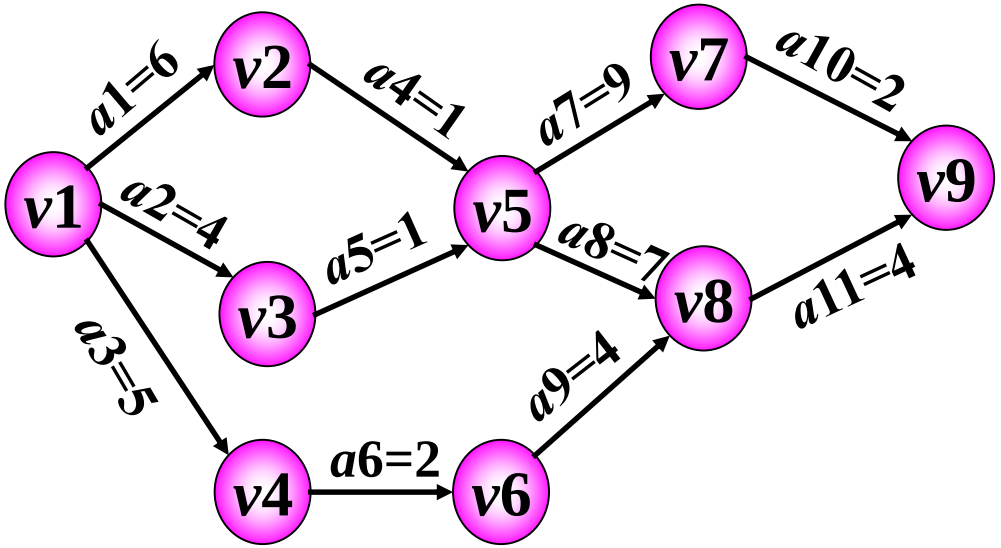
7.5.2 关键路径

把工程计划表示为有向图，用顶点表示事件，弧表示活动，弧的权表示活动持续时间。每个事件表示在它之前的活动已经完成，在它之后的活动可以开始。称这种有向图为边表示活动的网，简称为 AOE (Activity On Edge) 网。

例：设一个工程有 11 项活动，9 个事件。

事件 *v*1 — 表示整个工程开始（源点）

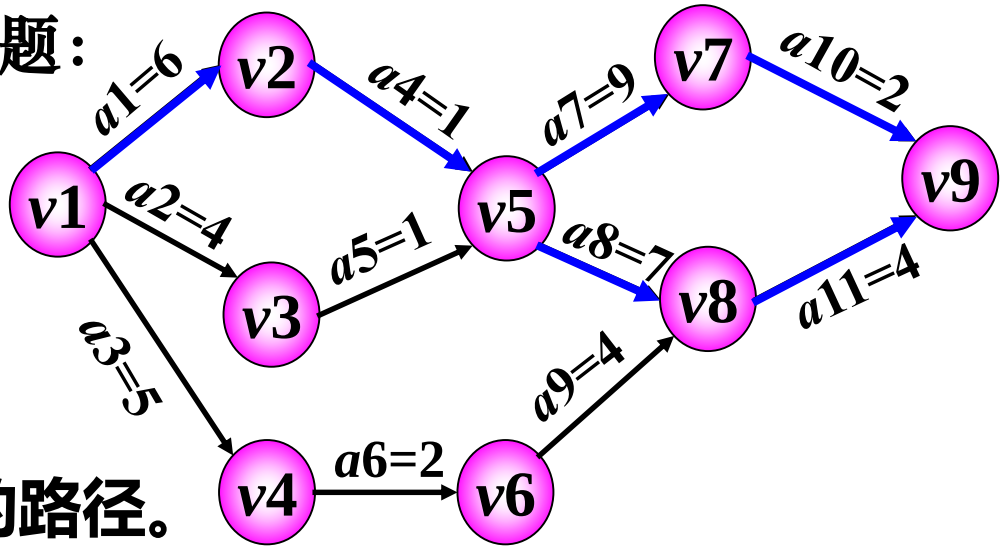
事件 *v*9 — 表示整个工程结束（汇点）



对AOE网，我们关心两个问题：

(1) 完成整项工程至少需要多少时间？

(2) 哪些活动是影响工程进度的关键？



关键路径 — 路径长度最长的路径。

路径长度 — 路径上各活动持续时间之和。

ve(*j*) — 表示事件 *vj* 的最早发生时间。 *ve*(3) = 4

vl(*j*) — 表示事件 *vj* 的最迟发生时间。 *vl*(3) = 6

e(*i*) — 表示活动 *ai* 的最早开始时间。 *e*(5) = 4

l(*i*) — 表示活动 *ai* 的最迟开始时间。 *l*(5) = 6

l(*i*) - *e*(*i*) — 表示完成活动 *ai* 的时间余量。 *l*(5) - *e*(5) = 2

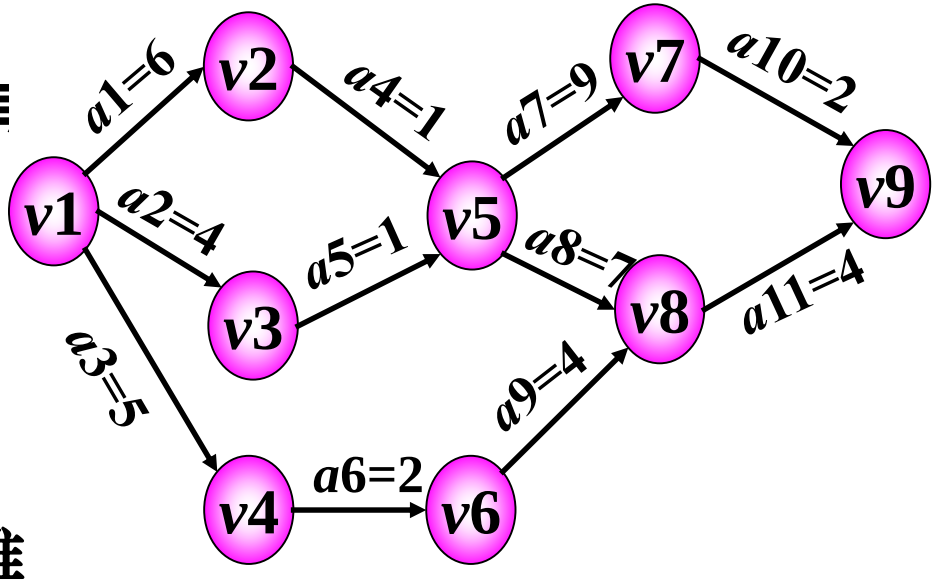
关键活动 — 关键路径上的活动，即 *l*(*i*) = *e*(*i*) 的活动。

如何找 $l(i) = e(i)$ 的关键活动?

设活动 ai 用弧 $<j, k>$ 表示，其

则有： (1) $e(i) = ve(j)$

(2) $l(i) = vl(k) - dut(<j, k>)$



如何求 $ve(j)$ 和 $vl(j)$?

(1) 从 $ve(1) = 0$ 开始向前递推

$$ve(j) = \underset{i}{Max}\{ve(i) + dut(<i, j>)\},\ <i, j> \in T,\ 2 \le j \le n$$

其中T 是所有以j 为头的弧的集合。

(2) 从 $vl(n) = ve(n)$ 开始向后递推

$$vl(i) = \underset{j}{Min}\{vl(j) - dut(<i, j>)\},\ <i, j> \in S,\ 1 \le i \le n-1$$

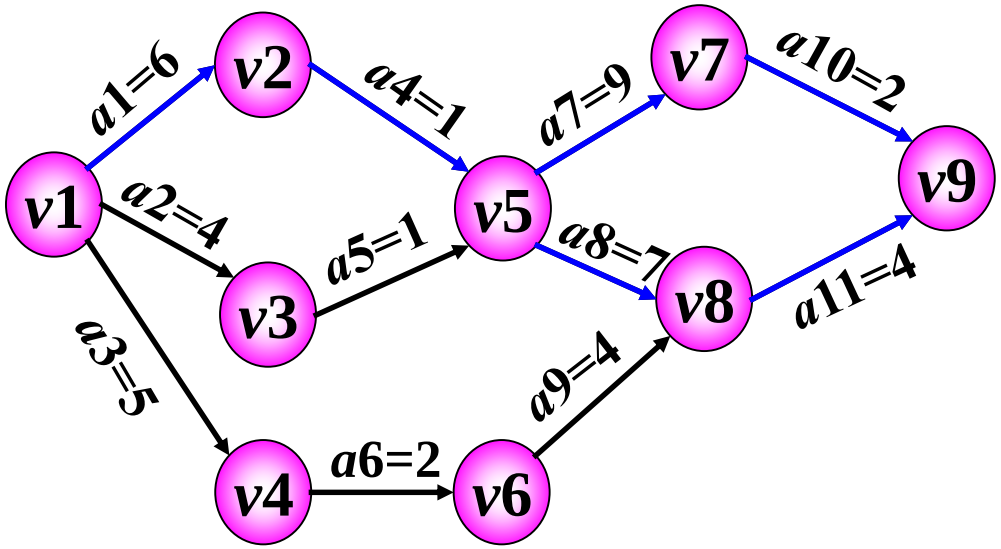
其中S 是所有以i 为尾的弧的集合。

求关键路径步骤：

求 *ve*(*i*)、*vl*(*j*)

求 *e*(*i*)、*l*(*i*)

计算 *l*(*i*) - *e*(*i*)



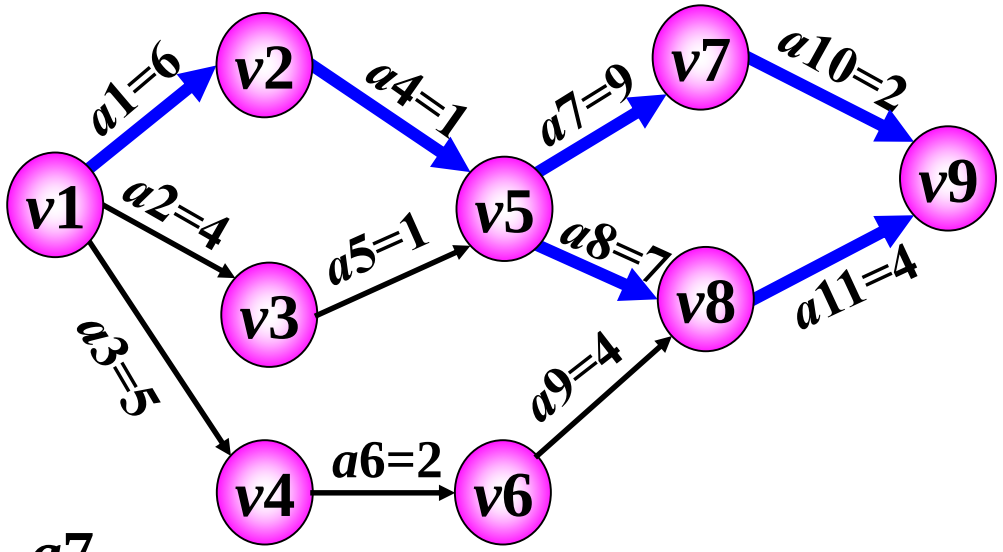
活动	*e*	*l*	*l* – *e*	
*a*1	0	0	0	✓
*a*2	0	2	2	
*a*3	0	3	3	
*a*4	6	6	0	✓
*a*5	4	6	2	
*a*6	5	8	3	
*a*7	7	7	0	✓
*a*8	7	7	0	✓
*a*9	7	10	3	
*a*10	16	16	0	✓
*a*11	14	14	0	✓

顶点	*ve*	*vl*
*v*1	0	0
*v*2	6	6
*v*3	4	6
*v*4	5	8
*v*5	7	7
*v*6	7	10
*v*7	16	16
*v*8	14	14
*v*9	18	18

关键路径的讨论

1、若网中有几条关键路径，则需加快同时在几条关键路径上的关键活动。

如：$a11$、$a10$、$a8$、$a7$。

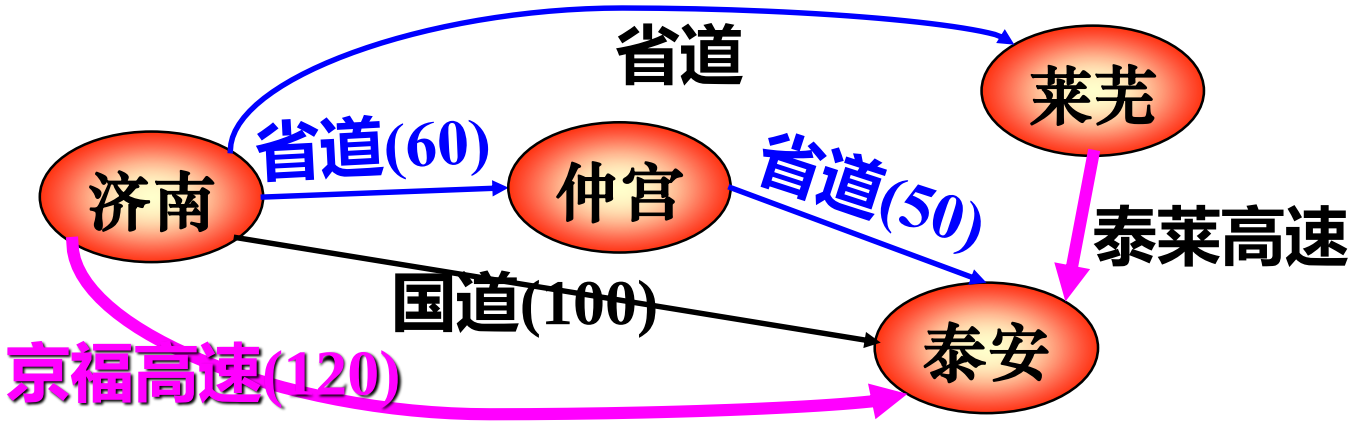


2、如果一个活动处于所有的关键路径上，则提高这个活动的速度，就能缩短整个工程的完成时间。如：a1、a4。

3、处于所有关键路径上的活动完成时间不能缩短太多，否则会使原关键路径变成非关键路径。这时必须重新寻找关键路径。如：$a1$由 6 天变成 3 天，就会改变关键路径。

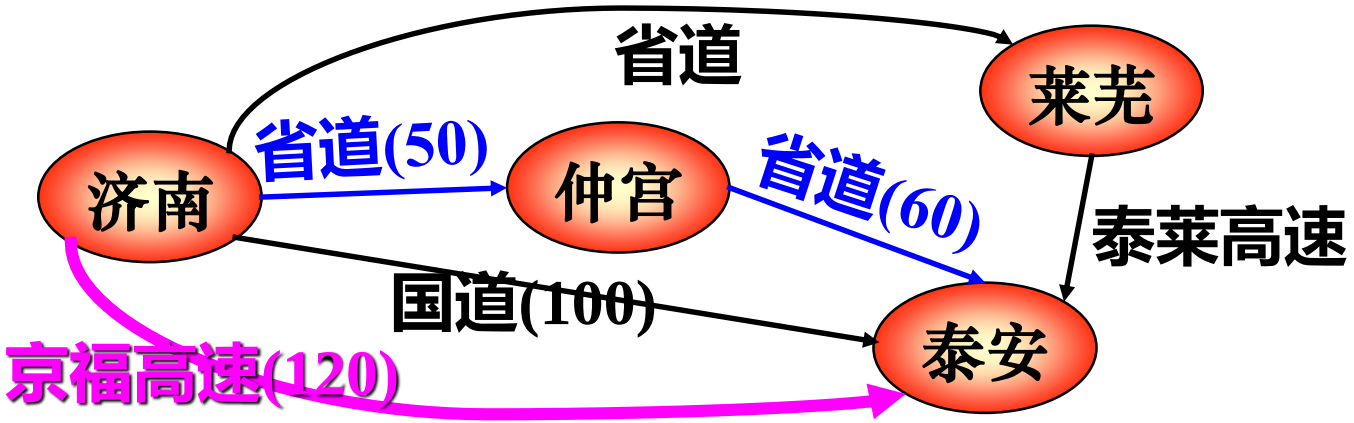
7.6 最短路径

典型用途：交通网络的问题——从甲地到乙地之间是否有公路连通？在有多条通路的情况下，哪一条路最短？



交通网络用有向网来表示：顶点——表示城市，弧——表示两个城市有路连通，弧上的权值——表示两城市之间的距离、交通费或途中所花费的时间等。

如何能够使一个城市到另一个城市的运输时间最短或运费最省？这就是一个求两座城市间的最短路径问题。

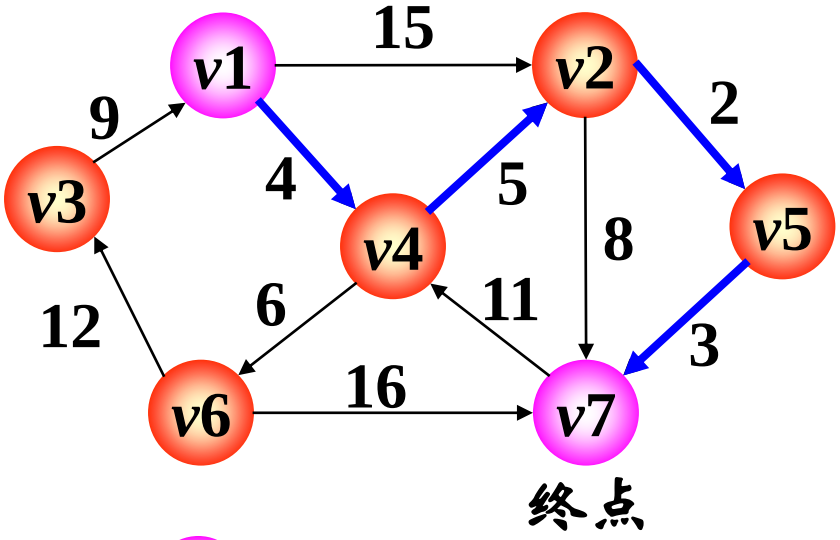


问题抽象：在有向网中A点（源点）到达B点（终点）的多条路径中，寻找一条各边权值之和最小的路径，即最短路径。

注

最短路径与最小生成树不同，路径上不一定包含 n 个顶点，也不一定包含 $n-1$ 条边。

例： 源点



从 v1 到 v7 的路径：

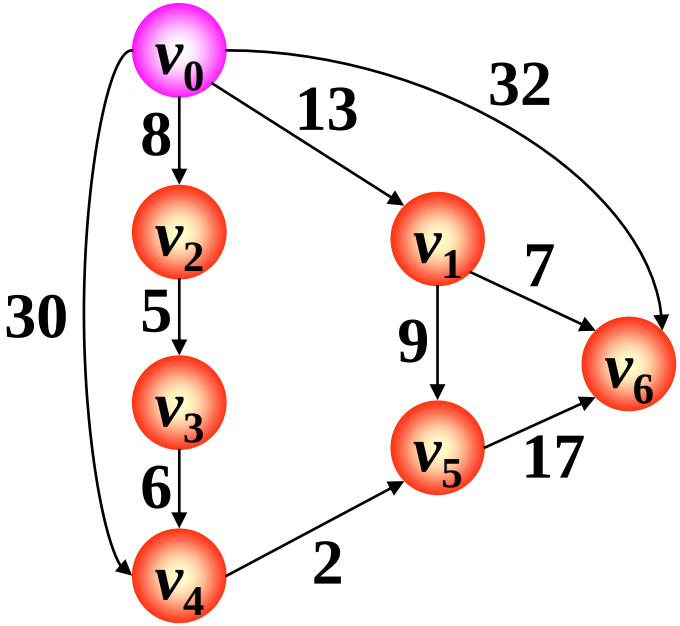
v1、v2、v5、v7： 20

v1、v4、v2、v5、v7： 14

v1、v2、v7： 23

v1、v4、v2、v7： 17

v1、v4、v6、v7： 24



最短路径	长度
(v_0, v_1)	13
(v_0, v_2)	8
(v_0, v_2, v_3)	13
(v_0, v_2, v_3, v_4)	19
$(v_0, v_2, v_3, v_4, v_5)$	21
(v_0, v_1, v_6)	20

两种最常见的最短路径问题

- 单源点最短路径：从某个源点到其余各顶点的最短路径
- 所有顶点间的最短路径：每对顶点间的最短路径

7.6.1 单源点最短路径（从某个源点到其余各顶点的最短路径）

怎样求单源点的最短路径呢？

1、将源点到终点的所有路径都列出来，

然后在其中选最短的一条。 穷举法

缺点：当路径特别多时，特别麻烦；

没有规律可循。

2、迪杰斯特拉（Dijkstra）算法：

按路径长度递增次序产生各顶点的最短路径。

- 路径长度最短的最短路径的特点：

  在此路径上，必定只含一条弧 $<v_0, v_1>$，且其权值最小。

  由此，只要在所有从源点出发的弧中查找权值最小者。 $<a, c>$

- 下一条路径长度次短的最短路径的特点：

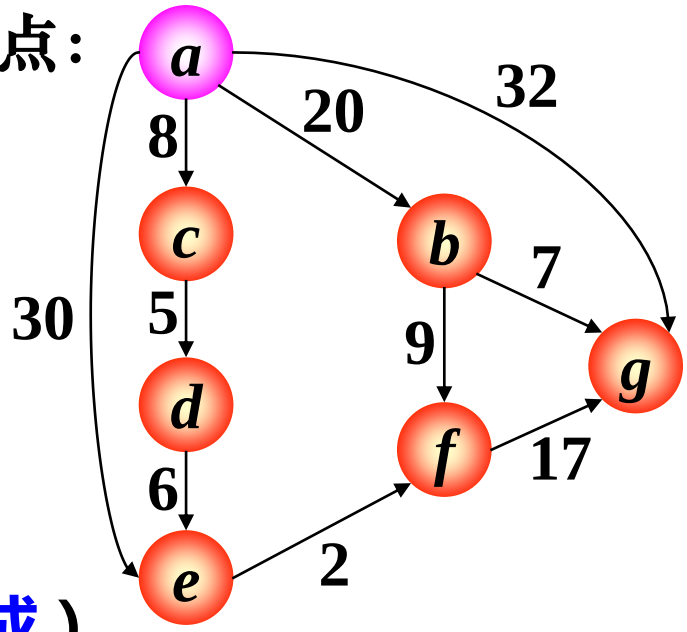
  它只可能有两种情况：

  1)、直接从源点到 v_2 $< v_0, v_2>$

  （只含一条弧）；

  2)、从源点经过顶点 v_1，再到达 v_2

  $< v_0, v_1>, < v_1, v_2>$（由两条弧组成）。



$<a, c>$, $<c, d>$

v_1 必为已求得的最短路径上的顶点

- 再下一条路径长度次短的最短路径的特点：

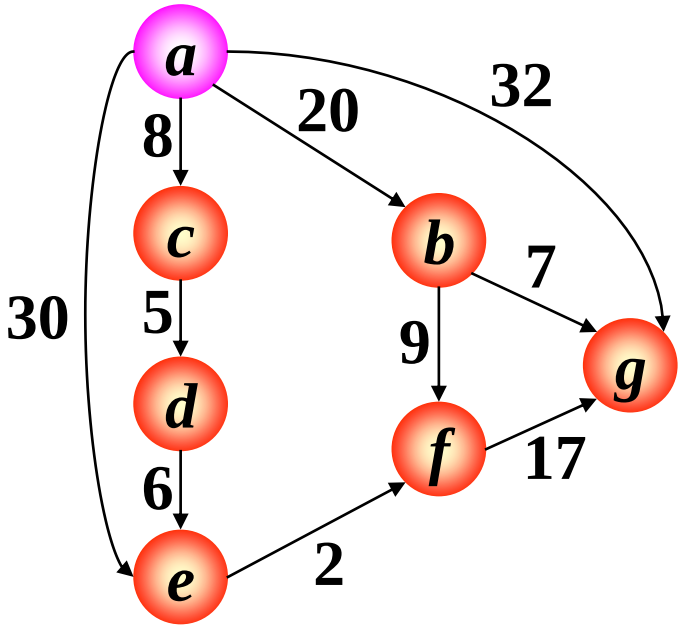
可能有四种情况：

1)、直接从源点到 v_3 $<v_0, v_3>$（由一条弧组成）；

2)、从源点经过顶点 v_1，再到达 v_3 $<v_0, v_1>$, $<v_1, v_3>$（由两条弧组成）；

3)、从源点经过顶点 v_2，再到达 v_3 $<v_0, v_2>$, $<v_2, v_3>$（由两条弧组成）；

4)、从源点经过顶点 v_1, v_2，再到达 v_3 $<v_0, v_1>$, $<v_1, v_2>$, $<v_2, v_3>$（由三条弧组成）；



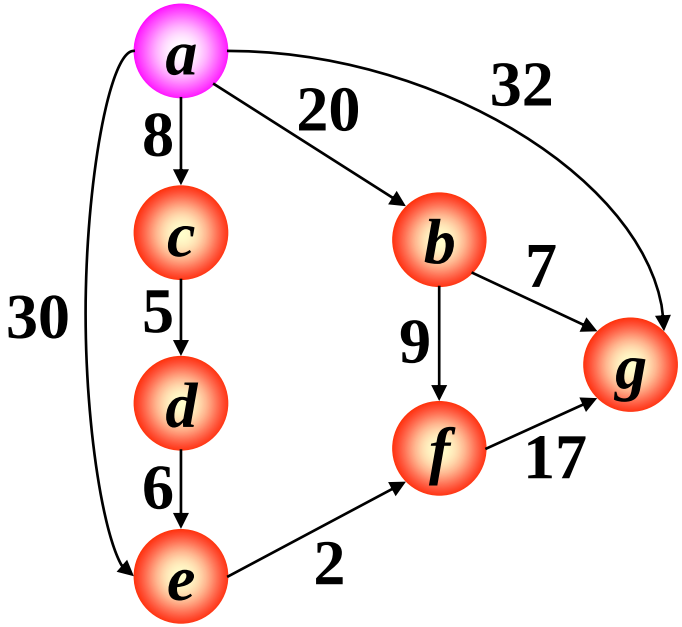
<a, c>，<c, d>，<d, e>

v_1，v_2 必为已求得的最短路径上的顶点

- 其余最短路径的特点：

  1)、直接从源点到 v_i $<v_0, v_i>$（只含一条弧）；

  2)、从源点经过已求得的最短路径上的顶点，再到达 v_i（含有多条弧）。



迪杰斯特拉 (Dijkstra) 算法：按路径长度递增次序产生最短路径

1、把 V 分成两组：

(1) S：已求出最短路径的顶点的集合。

(2) V - S = T：尚未确定最短路径的顶点集合。

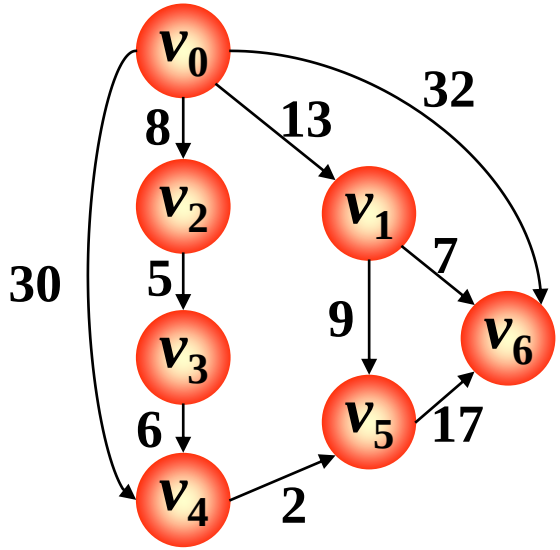
2、将 T 中顶点按最短路径递增的次序加入到 S 中，

保证：(1) 从源点 v_0 到 S 中各顶点的最短路径长度都不大于从 v_0 到 T 中任何顶点的最短路径长度。

(2) 每个顶点对应一个**距离值**：

S 中顶点：从 v_0 到此顶点的最短路径长度。

T 中顶点：从 v_0 到此顶点的只包括 S 中顶点作中间顶点的最短路径长度。



Dijkstra 算法步骤： 初始时令 S={v_0}，T={其余顶点}。

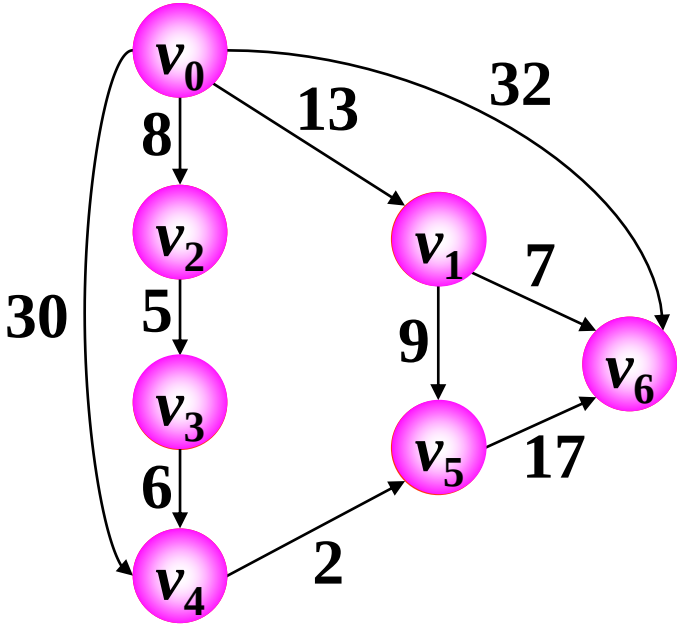
T 中顶点对应的距离值用辅助数组 D 存放。

$D[i]$ 初值：若 $<v_0, v_i>$ 存在，则为其权值；否则为∞。

从 T 中选取一个其距离值最小的顶点 v_j，加入 S。 $D[j]=\mathrm{Min}_i\{D[i]\mid v_i\in T\}$

对 T 中顶点的距离值进行修改：若加进 v_j 作中间顶点，从 v_0 到 v_i 的距离值比不加 v_j 的路径要短，则修改此距离值。

重复上述步骤，直到 S = V 为止。



终点	从 v_0 到各终点的最短路径及长度					
	i =1	i =2	i =3	i =4	i =5	i =6
v_1	13	13				
v_2	8					
v_3	∞	13	13			
v_4	30	30	30	19		
v_5	∞	∞	22	22	21	21
v_6	32	32	20	20	20	
v_j	v_2	v_1	v_3	v_4	v_6	v_5
s	8	13	8+5	8+5+6	13+7	8+5 +6+2

7.6.2 每一对顶点之间的最短路径

方法一：每次以一个顶点为源点，

重复执行 Dijkstra 算法 n 次。

方法二：弗洛伊德 (Floyd) 算法

算法思想：逐个顶点试探，从 v_i 到 v_j 的所有可能存在的路径中，选出一条长度最短的路径。

若$<v_i, v_j>$存在，则存在路径 $\{v_i, v_j\}$ // 路径中不含其它顶点

若 $<v_i, v_0>$, $<v_0, v_j>$ 存在，则存在路径 $\{v_i, v_0, v_j\}$

// 路径中所含顶点序号不大于 0

若$\{v_i, \ldots, v_1\}$, $\{v_1, \ldots, v_j\}$ 存在，则存在路径 $\{v_i, \ldots, v_1, \ldots, v_j\}$

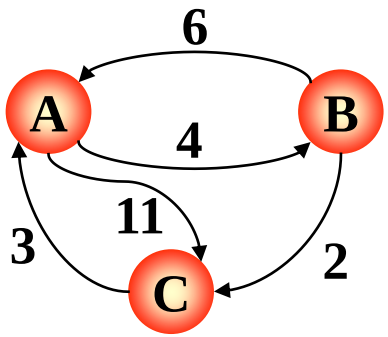
// 路径中所含顶点序号不大于 1

…… ……

求最短路径步骤：

初始时设置一个 n 阶方阵，令其对角线元素为 0，若存在弧$<v_i,v_j>$，则对应元素为权值；否则为∞。

逐步试着在原直接路径中增加中间顶点，若加入中间顶点后路径变短，则修改之；否则，维持原值。所有顶点试探完毕，算法结束。



初始：$\begin{pmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & \infty & 0 \end{pmatrix}$ 路径：

	AB	AC
BA		BC
CA		

加入A：$\begin{pmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & 7 & 0 \end{pmatrix}$ 路径：

	AB	AC
BA		BC
CA	CAB	

加入B：$\begin{pmatrix} 0 & 4 & 6 \\ 6 & 0 & 2 \\ 3 & 7 & 0 \end{pmatrix}$ 路径：

	AB	ABC
BA		BC
CA	CAB	

加入C：$\begin{pmatrix} 0 & 4 & 6 \\ 5 & 0 & 2 \\ 3 & 7 & 0 \end{pmatrix}$ 路径：

	AB	ABC
BCA		BC
CA	CAB	

教学要求

1、了解图的基本概念，掌握图的邻接矩阵、邻接表这两种存储结构及其构造方法；

2、熟练掌握图的两种遍历方法；

3、熟练掌握构造最小生成树的方法，并理解算法；

4、掌握 AOV 网的拓扑排序方法，并理解算法；

5、掌握求解关键路径的方法；

6、理解用 Dijkstra 方法求解单源点最短路径问题。