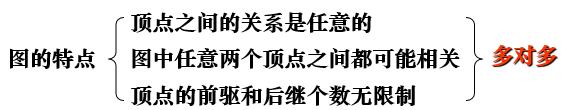
# 图的基本概念

**图（Graph）是一种非线性结构。**



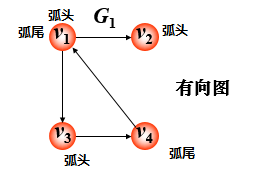
## 图的定义

**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。**

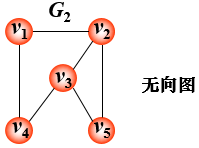
## 图的基本术语

**顶点**：图中的数据元素。

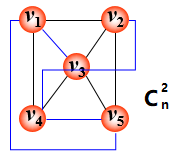
**弧**：若 <*v*, *w*>∈*VR*，则 <*v*, *w*> 表示从 *v* 到 *w* 的一条弧，且称 *v* 为弧尾，称 *w* 为弧头，此时的图称为**有向图**。



**边**：若 <*v*, *w*>∈*VR* 必有<*w*, *v*>∈*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**。

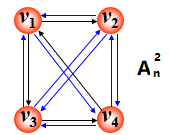


**无向图中边的取值范围：0≤*e*≤*n*(*n*-1)/2**。（用 *n* 表示图中顶点数目，用 *e* 表示边的数目。且不考虑顶点到其自身的边。）



**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≤*e*≤*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。）



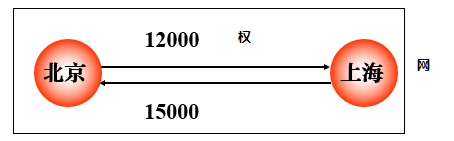
**有向完全图：**有 *n* (*n* - 1) 条弧的有向图（即：每两个顶点之间都存在着方向相反的两条弧）称为有向完全图。

**稀疏图：**含有很少条边或弧的图。

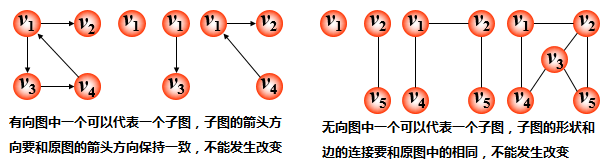
**稠密图：**含有很多条边或弧的接近完全图的图。

**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。

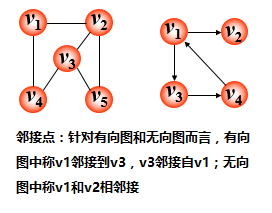


**子图**：如果图 *G* = (*V*, *E*) 和 *G*´= (*V*´, *E*´)，满足：*V*´⊆ *V* 且 *E*´⊆ *E*，则称 *G*´为*G* 的子图。



**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。

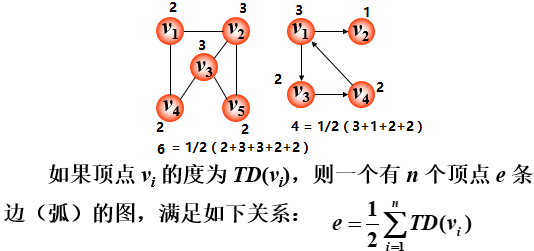


**度：**无向图中顶点 *v* 的度是和 *v* **相关联**的边的数目，记为：*TD*(*v*)。

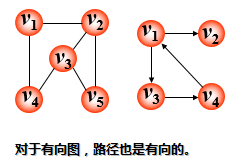
**入度：**有向图中以顶点 *v* 为头的弧的数目称为 *v* 的入度，记为：*ID*(*v*)。

**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：有向图中的度是入度和出度之和**，即：*TD*(*v*) = *ID*(*v*) + *OD*(*v*)。



**路径**：从顶点 *v* 到 *v*´ 的路径是一个顶点序列 (*v* = *vi,* 0, *vi,* 1, …, *vi, m*= *v*´)，满足 (*vi*, *j*-1, *vi*, *j*)∈*VR* 或 <*vi*, *j*-1, *vi*, *j* >∈*VR，*(1 ≤ *j* ≤ *m*)。



**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**。

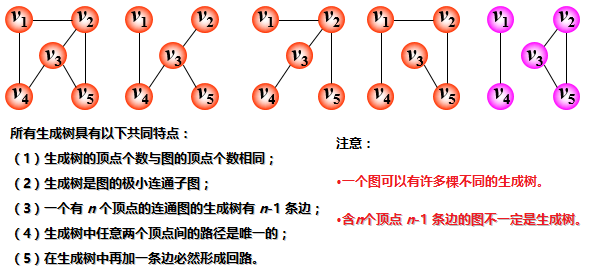
非连通图：**有 *n* 个顶点和小于 *n*-1 条边的图**。

**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。**



# 图的存储结构

## 数组表示法（邻接矩阵表示法）（重要）

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。**

**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, AG, AN} 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（比如相关的内容表示距离或者是人口流动的数量）

}ArcCell, AdjMatrix[MAX\_VERTEX\_NUM][MAX\_VERTEX\_NUM];

typedef struct {

VertexType vexs[MAX\_VERTEX\_NUM]; // 顶点向量

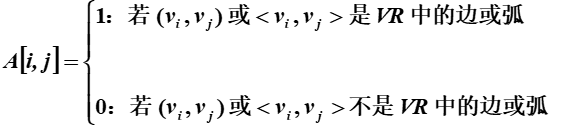
AdjMatrix arcs; // 邻接矩阵

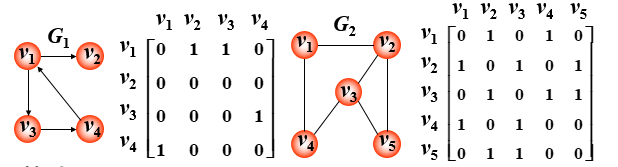
int vexnum, arcnum; // 图的当前顶点数和弧(边)数

GraphKind kind; // 图的种类标志

} MGraph;

**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, …, *vn*}，则 *G* 的邻接矩阵是具有如下性质的 ***n* 阶方阵**：





特点：

（1）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。

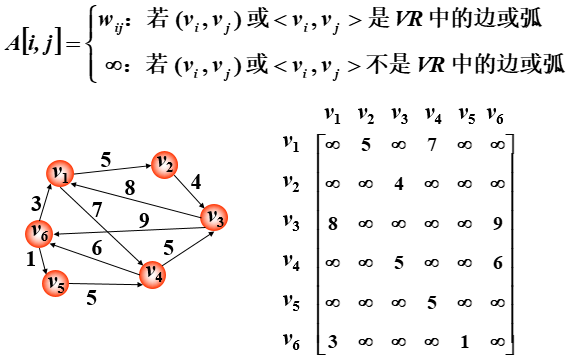
（2）有向图的邻接矩阵不一定对称；有 *n* 个顶点的有向图所需存储空间为*n*²，**用于稀疏图时空间浪费严重**。

**（3）无向图中顶点 *vi* 的度 *TD*(*vi*) 是邻接矩阵中第 *i* 行 1** **的个数。**

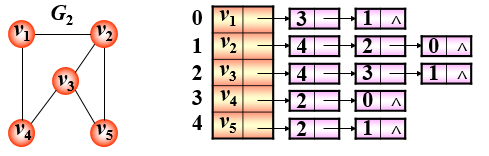


**常见题型：给出邻接矩阵表示法的存储结构，基于此存储结构求度。**

**网的邻接矩阵可定义为：**



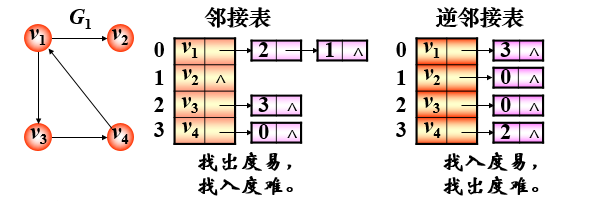
## 邻接表（类似于树的孩子链表表示法）

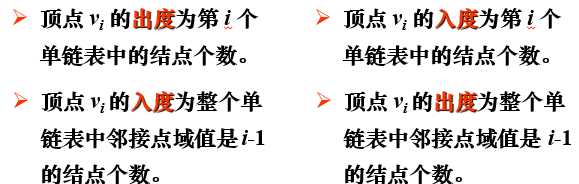




特点：

* 若无向图中有 ***n*** 个顶点、***e*** 条边，则其邻接表需 ***n*** 个头结点和 **2*e*** 个表结点。适宜存储稀疏图。
* 无向图中顶点 *vi* 的**度为第 *i* 个单链表中的结点数**。





**图的邻接表存储表示：**

#define MAX\_VERTEX\_NUM 20

bool visited[MAX\_VERTEX\_NUM];

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

} 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 {

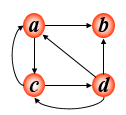
AdjList vertices; \_

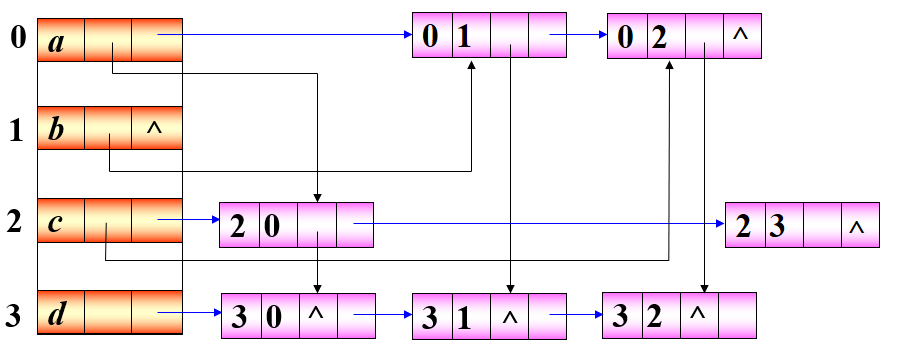
int vexnum, arcnum; // 图的当前顶点数和弧数

} ALGraph;

## 十字链表

只能针对**有向图**而言





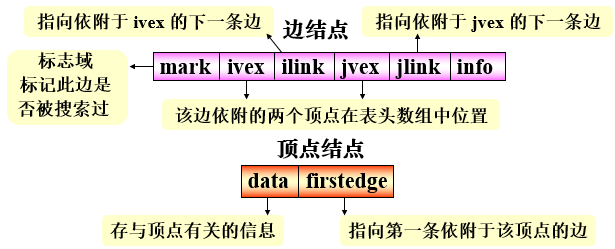
## 邻接多重表（无向图的另一种链式存储结构）

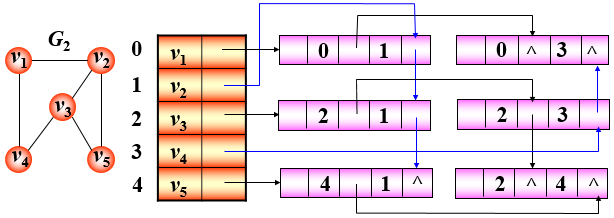
只能针对**无向图**而言

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：**





# 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。

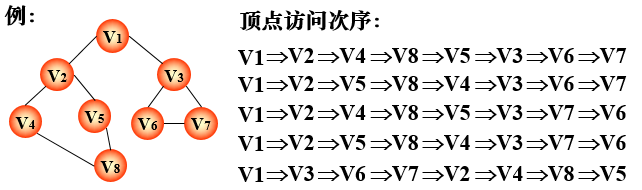
## 深度优先遍历（DFS）

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。

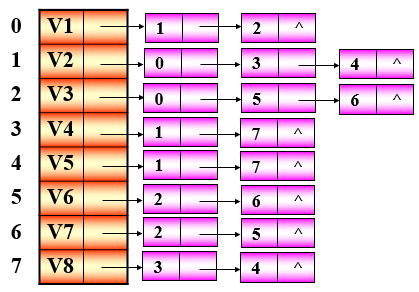




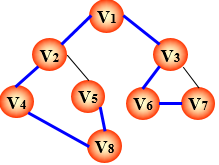
**如何判别V的邻接点是否被访问？**

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。

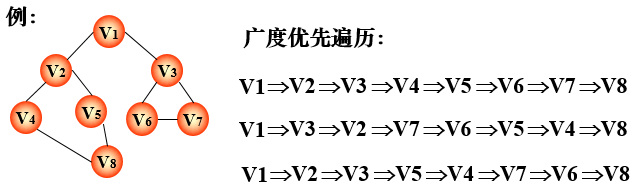


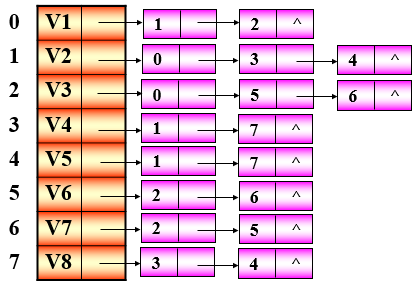
**根据上述的存储结构示意图，画出图的结构并且能够写出深度优先遍历序列。**



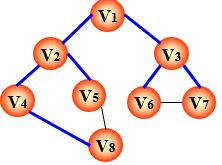
## 广度优先遍历（BFS）

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 V*i*1, V*i*2, …, V*in* **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。





**根据上述的存储结构示意图，画出图的结构并且能够写出广度优先遍历序列。**





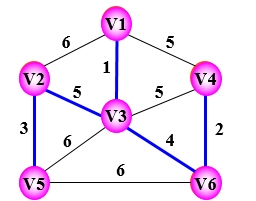
# 最小生成树

**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。

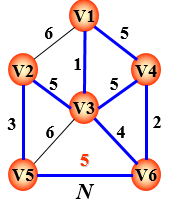
最小生成树是n个顶点，n-1条边。

## 构造最小生成树方法

**方法一：普里姆 (Prim) 算法。**

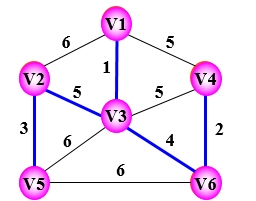


**方法二：克鲁斯卡尔 (Kruskal) 算法。**



**最小生成树可能不惟一 （注意用克鲁斯卡尔算法不能形成环路）**

## 模拟内存的变化



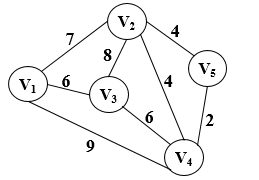
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** | **U** | **V-U** | **k** |
| **V1**  **6** | **V1**  **1** | **V1**  **5** |  |  | **{v1}** | **{v2,v3,v4,v5,v6}** | **2** |
| **V3**  **5** | **0** | **V1**  **5** | **V3**  **6** | **V3**  **4** | **{v1,v3}** | **{v2,v4,v5,v6}** | **5** |
| **V3**  **5** | **0** | **V6**  **2** | **V3**  **6** | **0** | **{ v1,v3,v6}** | **{v2,v4,v5 }** | **3** |
| **V3**  **5** | **0** | **0** | **V3**  **6** | **0** | **{ v1,v3,v6,v4}** | **{v2,v5}** | **1** |
| **0** | **0** | **0** | **V2**  **3** | **0** | **{ v1,v3,v6,v4 v2 }** | **{v5}** | **4** |
| **0** | **0** | **0** |  | **0** | **{v1v2,v3,v4,v5,v6}** | **{v2,v3,v4,v5,v6}** |  |

## 练习

1. *n* 个顶点的连通图至少 ( n – 1 ) 条边。

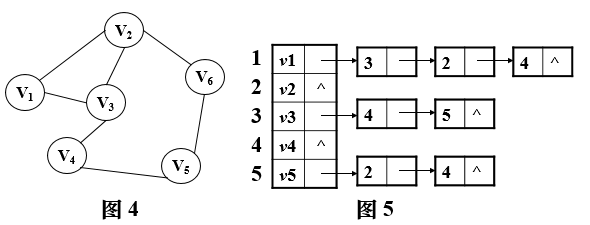
2. 在一个无向图的邻接表中，若表结点的个数是 *m*，则图中边的条数是 (m/2 ) 条。

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。

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 (C) 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 (B) 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 (C) 条边。   
 （A）*n* （B）*n*(*n*-1) （C）*n*(*n*-1)/2 （D）2*n*   
4. 具有 4 个顶点的无向完全图有 (A) 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 (A) 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 (C) 条边。   
 （A）*n* （B）*n* + 1 （C）*n* – 1 （D）*n*/2

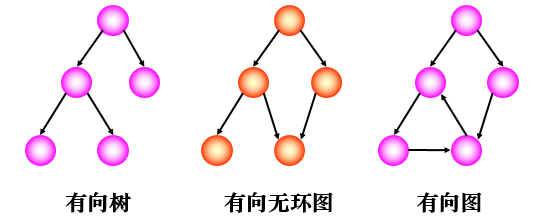
1. 对于一个具有 *n* 个顶点的无向图，若采用邻接矩阵表示，则该矩阵的大小为 (D)。   
   （A）*n* （B）(*n*-1)2 （C）*n* -1 （D）*n*2
2. 对于一个具有 *n* 个顶点和 *e* 条边的无向图，若采用邻接表表示，则表头数 组的大小为 (A)，所有邻接表中表结点的总数是 (C)。   
   ①（A）*n* （B）*n*+1 （C）*n*-1 （D）*n*+*e*   
   ②（A）*e*/2 （B）*e* （C）2*e* （D）*n*+*e*

9. 图的深度优先遍历算法类似于树的（A）。  
（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（C）。  
（A）先根遍历 （B）后根遍历 （C）按层遍历

# 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。



## 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。

**AOV 网的特点：**

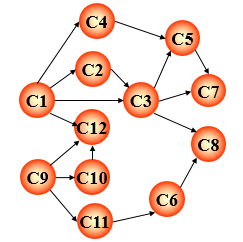
若从 *i* 到 *j* 有一条**有向路径**，则 *i* 是 *j* 的前驱；*j* 是 *i* 的后继。

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？**

**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。

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

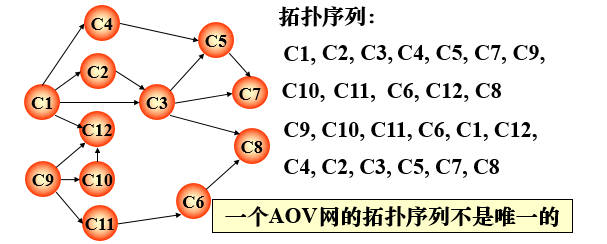
弧 <*i*, *j*> 存在，则在这个序列中，*i* 一定排在 *j*

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。**

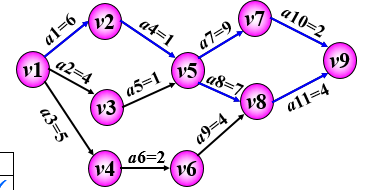
**拓扑排序的方法：**

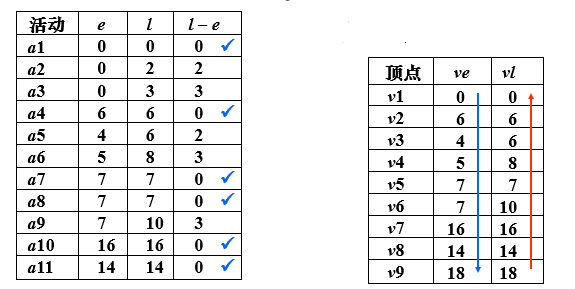
* 在有向图中选一个没有前驱的顶点且输出之。
* 从图中删除该顶点和所有以它为尾的弧。
* 重复上述两步，直至全部顶点均已输出；或者当图中不存在无前驱的顶点为止。



## 关键路径

**求关键路径步骤：求 *ve*(*i*)、*vl*(*j*) 求 *e*(*i*)、*l*(*i*) 计算 *l*(*i*) - *e*(*i*)**





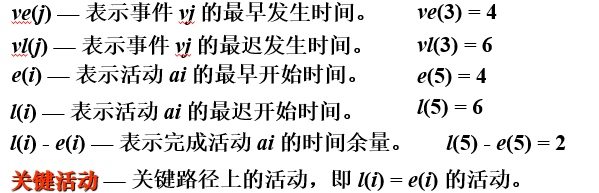
**对AOE网，我们关心两个问题：**

(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**。

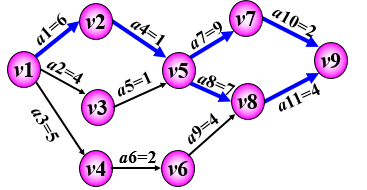


**如何找 *l*(*i*) = *e*(*i*) 的关键活动？**

**设活动 *ai* 用弧 <*j*, *k*> 表示，其持续时间记为：*dut*(<*j*, *k*>)**

**则有： (1) *e*(*i*) = *ve*(*j*)**

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

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：*a*11、*a*10、*a*8、*a*7。

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：*a*1由 6 天变成 3 天，就会改变关键路径。

# 最短路径

## 单源点最短路径（从某个源点到其余各顶点的最短路径）

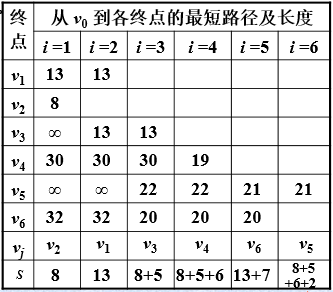
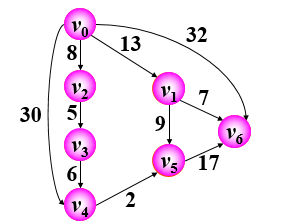
迪杰斯特拉（Dijkstra）算法：

按路径长度递增次序产生各顶点的最短路径。

**最短路径的特点：**

**1)、直接从源点到 *vi* <*v*0, *vi* >（只含一条弧）；（最开始的第一条弧即是在所有从源点出发的弧中查找权值最小者）**

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



## 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。对于方法一来说总的执行时间是***o（n3）***

方法二：

弗洛伊德 (Floyd) 算法

算法思想：逐个顶点试探，从 *vi* 到 *vj* 的所有可能存在的路径中，选出一条长度最短的路径。 这个算法的时间复杂度也是***o（n3）***

