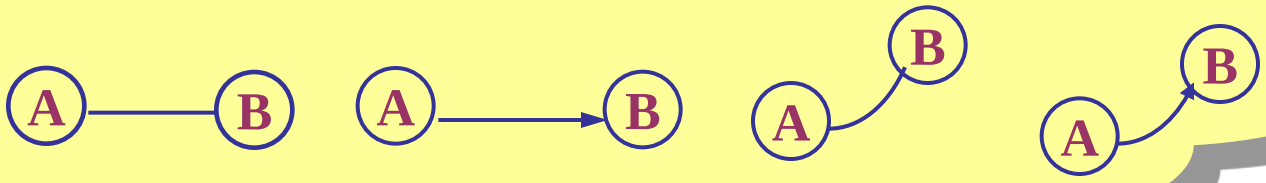
第八章

图

一个数据元素-----*顶点.*

数据元素(顶点)之间的关系-----*边(弧)*



本章讨论的主要内容：

8.1 图的基本概念
8.2 图的存储方法
8.3 图的遍历
8.4 最小生成树
8.5 最短路径问题
8.6 AOV网与拓扑排序
8.7 AOE网与关键路径

应用

8.1 图的基本概念

一. 图的定义

图是由顶点的非空有穷集合与顶点之间关系(边或弧)的集合构成的结构，通常表示为

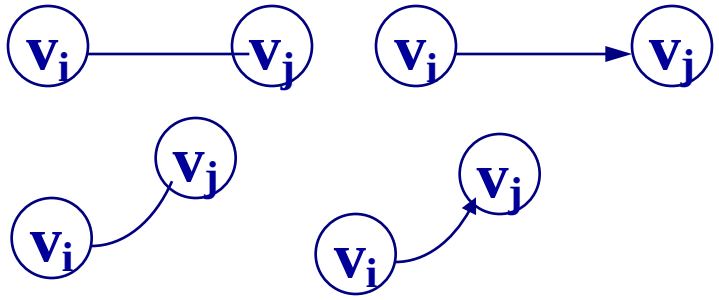
$$G = (V, E)$$

其中，V 为顶点集合，E 为关系(边或弧)的集合。

非空有穷集合

关于一条边或弧的表示方法:

(1). 用图形:



(2). 用符号:

(v_i, v_j) 或 $<v_i, v_j>$

顶点偶对

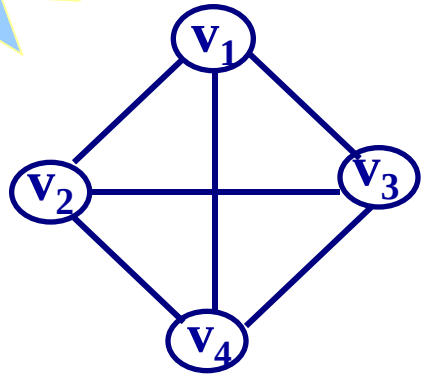
无向图

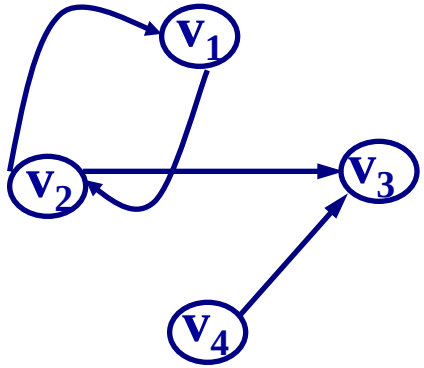
有向图

(3). 用语言:

- 顶点v_i与v_j是这条边的两个邻接点。
- 这条边依附于顶点v_i和顶点v_j。

例





$G_1=(V_1, E_1)$

其中

$V_1 = \{ v_1, v_2, v_3, v_4 \}$

$E_1 = \{ (v_1,v_2), (v_1,v_3), (v_1,v_4), (v_2,v_3), (v_2,v_4), (v_3,v_4) \}$

$G_2=(V_2, E_2)$

其中

$V_2 = \{ v_1, v_2, v_3, v_4 \}$

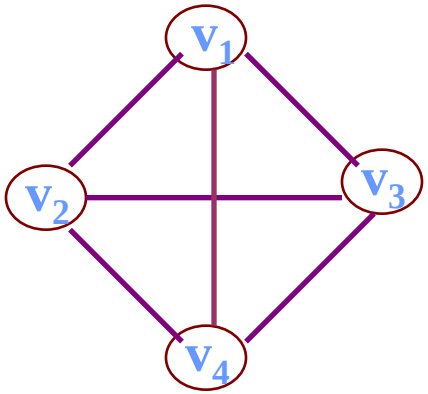
$E_2 = \{ \langle v_1,v_2\rangle, \langle v_2,v_1\rangle, \langle v_2,v_3\rangle, \langle v_4,v_3\rangle \}$

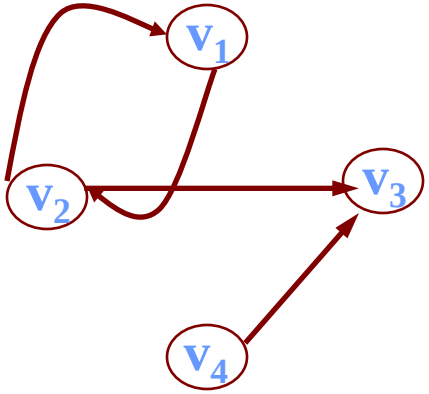
二. 图的分类

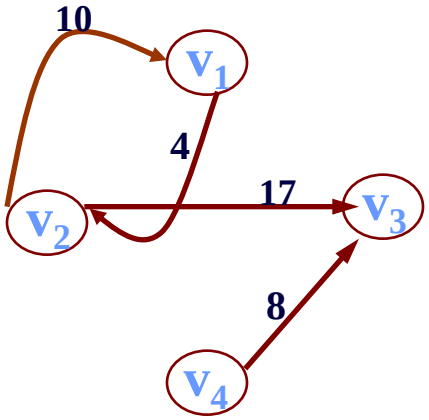
无向图　对于（v_i,v_j）$\in E$，必有（v_j,v_i）$\in E$，并且偶对中顶点的前后顺序无关。

有向图　若$<v_i,v_j>\in E$是顶点的有序偶对。

网(络)　与边有关的数据称为权，边上带权的图称为网络。







三. 名词术语

1. 顶点的度

依附于顶点v_i的边的数目，记为$TD(v_i)$.

对于有向图而言，有：

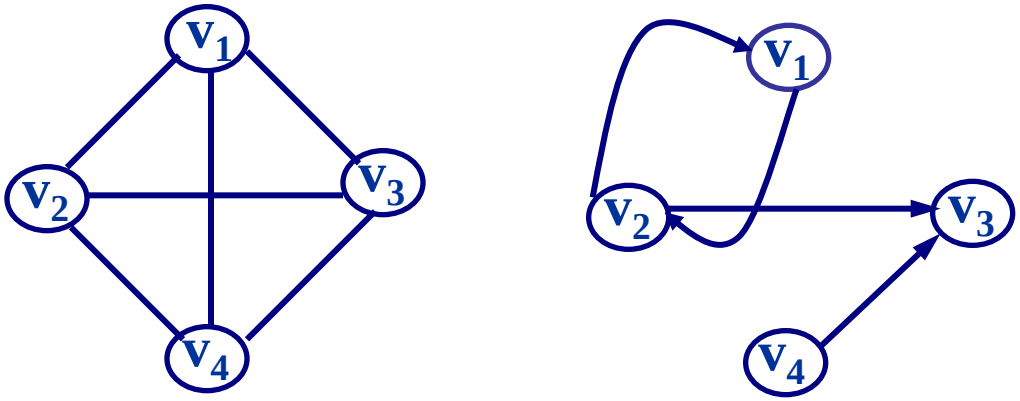
顶点的**出度**：

以顶点v_i为出发点的边的数目，记为$OD(v_i)$.

顶点的**入度**：

以顶点v_i为终止点的边的数目，记为$ID(v_i)$.

$$TD(v_i) = OD(v_i) + ID(v_i)$$



结论1 对于具有n个顶点, e条边的图, 有

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

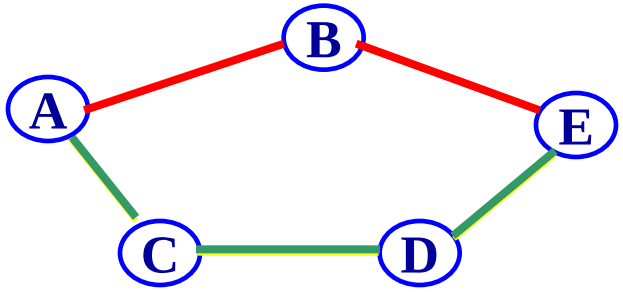
结论2 具有n个顶点的无向图最多有n(n-1)/2 条边。

结论3 具有n个顶点的有向图最多有n(n-1) 条边.

边的数目达到最大的图称为 ***完全图***。边的数目达到或接近最大的图称为 ***稠密图***, 否则, 称为 ***稀疏图***。

2. 路径

顶点v_x到v_y之间有路径$P(v_x, v_y)$的充分必要条件为：存在顶点序列 $v_x, v_{i1}, v_{i2}, \cdots, v_{im}, v_y$， 并且序列中相邻两个顶点构造的顶点偶对分别为图中的一条边。

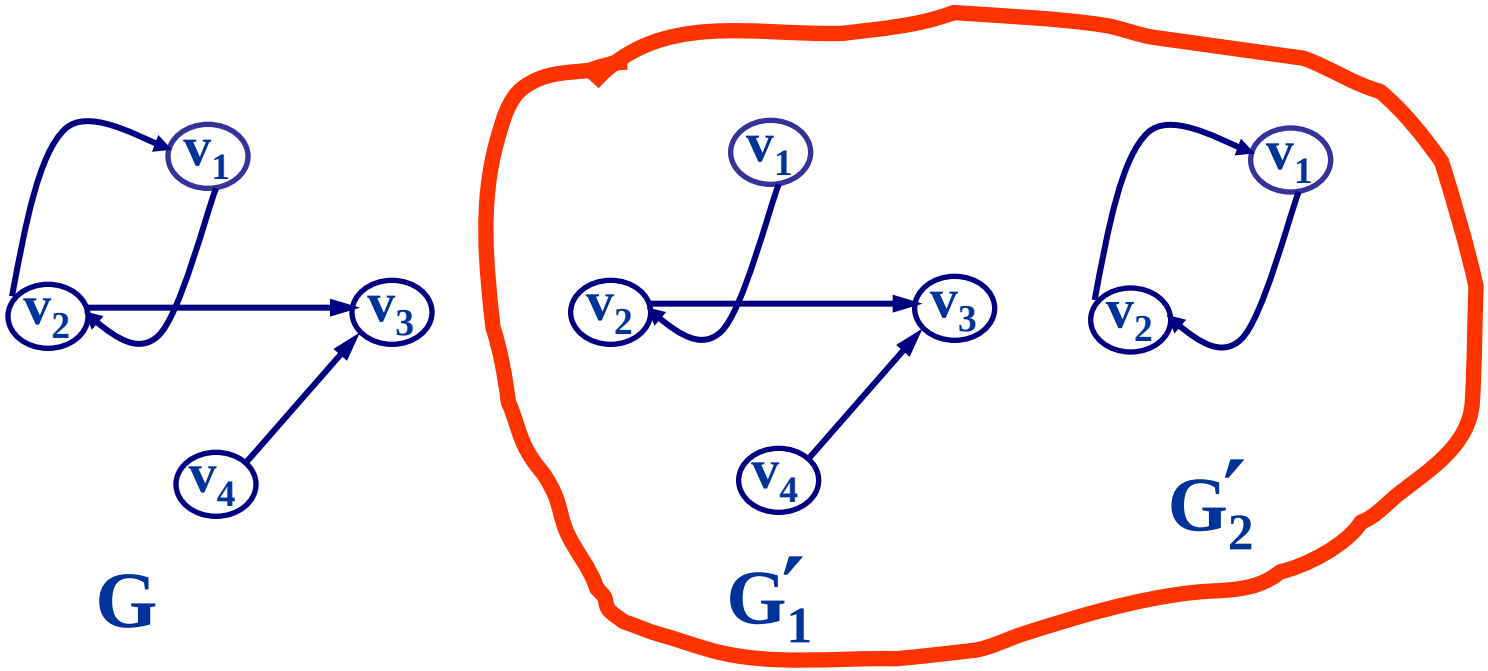


P(A, E): A, B, E （第1条路径）
A, C, D, E（第2条路径）

出发点与终止点相同的路径称为回路或环；顶点序列中顶点不重复出现的路径称为简单路径。不带权的图的路径长度是指路径上所经过的边的数目，带权的图的路径长度是指路径上经过的边上的权值之和。

3. 子图

对于图G=(V,E) 与 G′=(V′,E′)，若有V′⊆V, E′⊆E, 则称G′为G的一个子图。

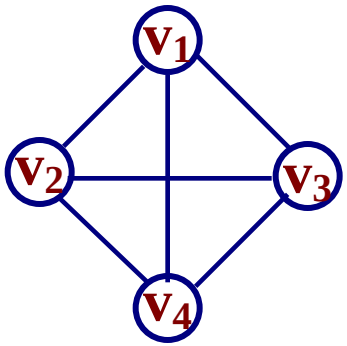


4. 图的连通

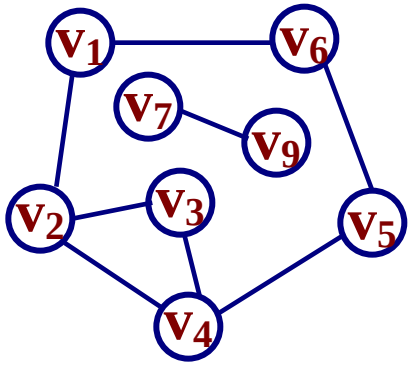
(1) 无向图的连通

无向图中顶点v_i到v_j有路径，则称顶点v_i与v_j是连通的。若无向图中任意两个顶点都连通，则称该无向图是连通的。

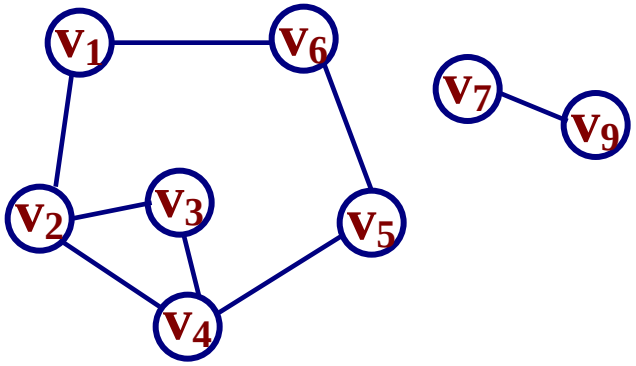
连通分量 —— 无向图中的极大连通子图。



连通图



非连通图



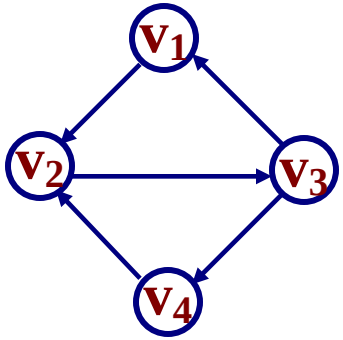
2个连通分量

(2) 有向图的连通

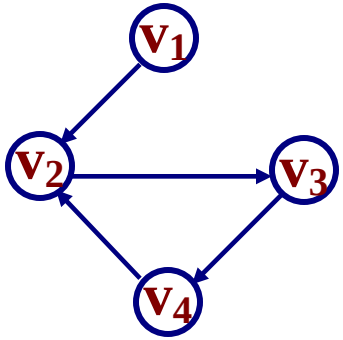
若有向图中顶点v_i到v_j有路径，并且顶点v_j到v_i也有路径，则称顶点v_i与v_j是连通的。

若有向图中任意两个顶点都连通，则称该有向图是强连通的。

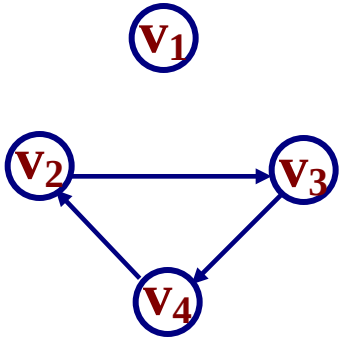
强连通分量 —— 有向图中的极大强连通子图。



(强)连通图



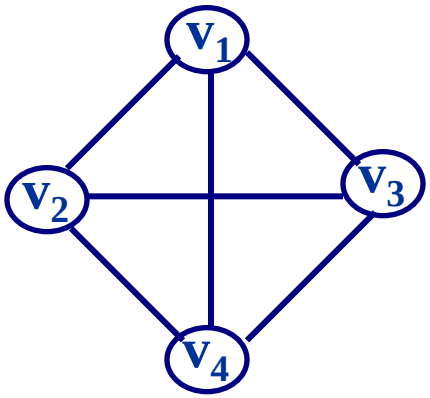
非连通图



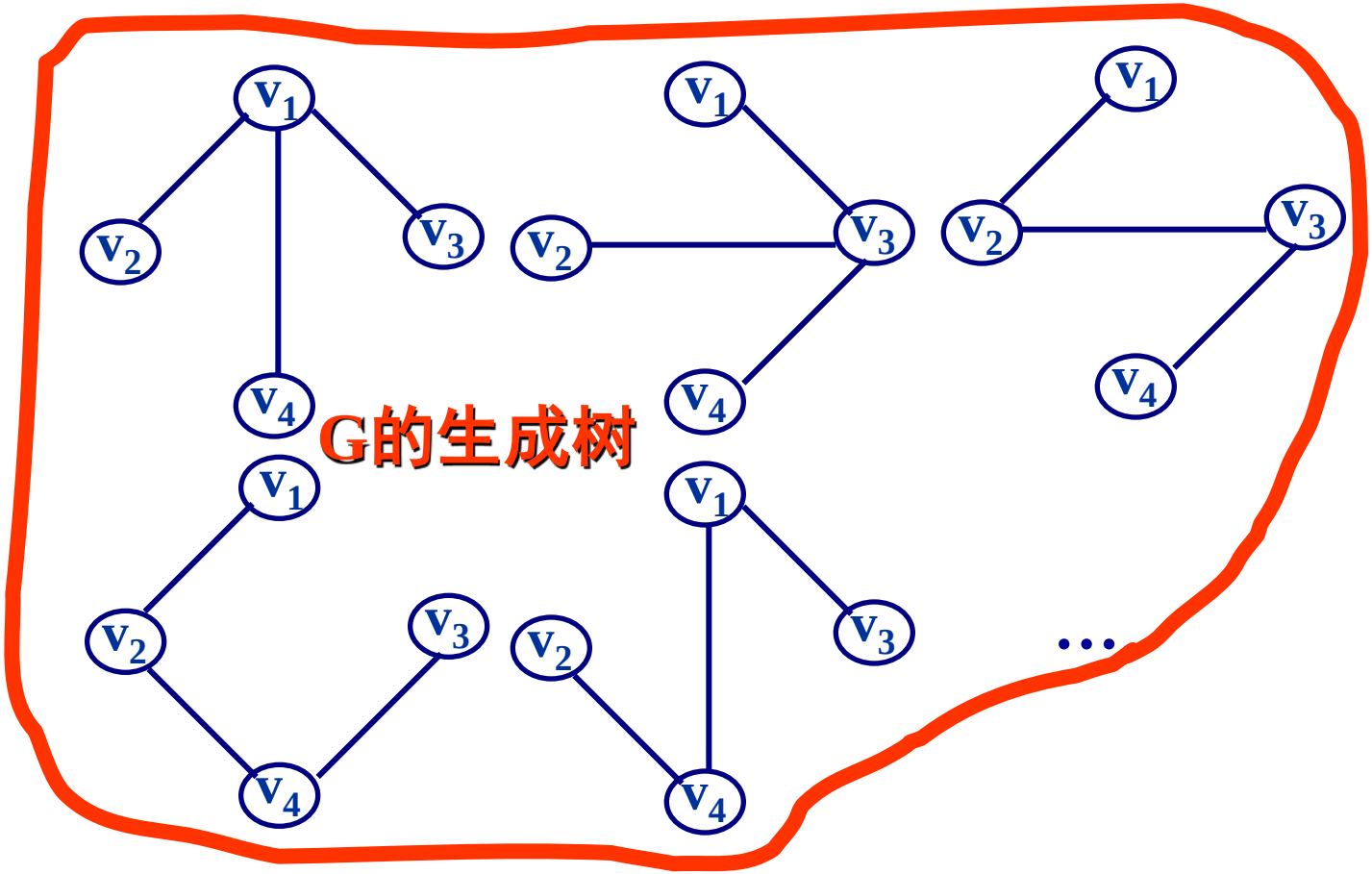
2个强连通分量

5. 生成树

包含具有n个顶点的连通图G的全部n个顶点，仅包含其n−1条边的极小连通子图称为G的一个生成树。

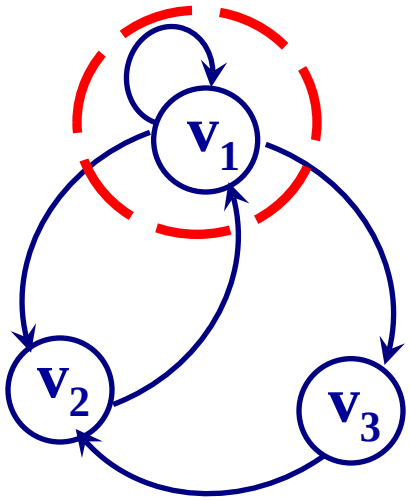


G

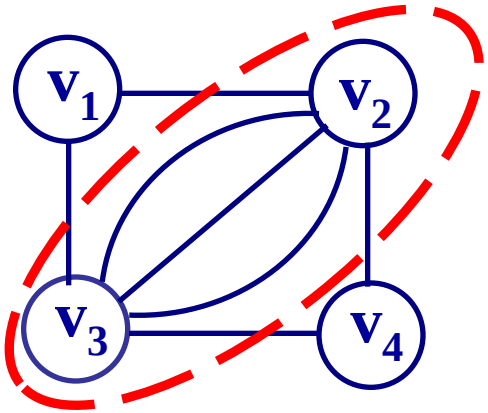


本章不讨论的图

1. 带自身环的图



2. 多重图



8.2 图的存储方法

对于一个图，需要存储的信息应该包括：

(1). 所有顶点的数据信息；

(2). 顶点之间关系（边或弧）的信息；

(3). 权的信息（对于网络）。

一. 邻接矩阵存储方法

数组存储方法

核心思想：采用两个数组存储一个图。

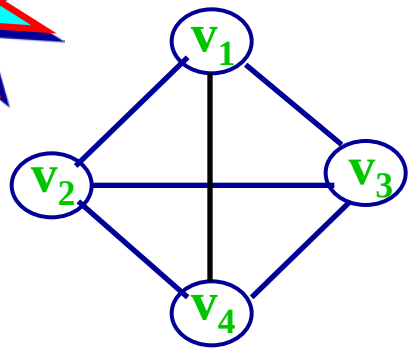
1. 定义一个一维数组VERTEX[0:n-1]存放图中所有顶点的数据信息(若顶点信息为1,2,3, …, 此数组可略)。
2. 定义一个二维数组A[0:n-1,0:n-1]存放图中所有顶点之间关系的信息(该数组被称为**邻接矩阵**), 有

$$A[i,j]=\begin{cases}1 & \text{当顶点}v_i\text{到顶点}v_j\text{有边时}\\ 0 & \text{当顶点}v_i\text{到顶点}v_j\text{无边时}\end{cases}$$

对于带权的图，有

$$A[i,j]=\begin{cases}w_{ij} & \text{当顶点}v_i\text{到}v_j\text{有边, 且边的权为}w_{ij}\\ \infty & \text{当顶点}v_i\text{到顶点}v_j\text{无边时}\end{cases}$$

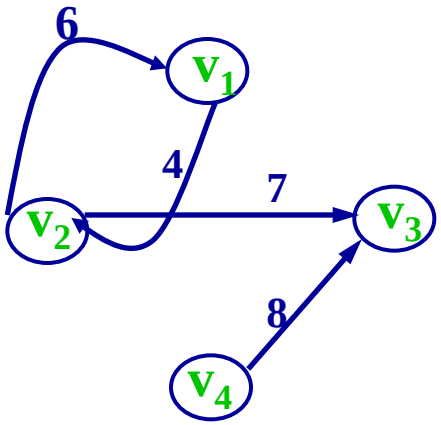
例



VERTEX1[0:3]

v_1
v_2
v_3
v_4

$$A_1 = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$



VERTEX2[0:3]

v_1
v_2
v_3
v_4

$$A_2 = \begin{bmatrix} \infty & 4 & \infty & \infty \\ 6 & \infty & 7 & \infty \\ \infty & \infty & \infty & \infty \\ \infty & \infty & 8 & \infty \end{bmatrix}$$

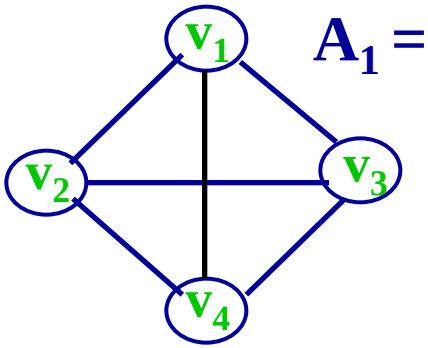
特点

(1). 无向图的邻接矩阵一定是一个对称矩阵。

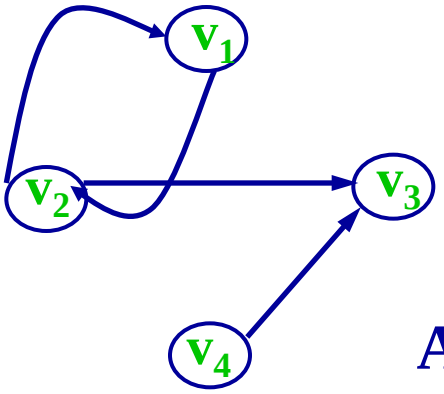
(2). 不带权的有向图的邻接矩阵一般是稀疏矩阵。

(3). 无向图的邻接矩阵的第i 行(或第i 列)非0 或非∞元素的个数为第i 个顶点的度数。

(4). 有向图的邻接矩阵的第i 行非0或非∞元素的个数为第i 个顶点的出度;第i 列非0或非∞元素的个数为第i 个顶点的入度。



$$A_1 = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$



$$A_2 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

二. 邻接表存储方法

核心思想:对具有n个顶点的图建立n个线性链表存储该图。

1. 每一个链表前面设置一个头结点,用来存放一个顶点的数据信息,称之为**顶点结点**。其构造为

vertex	link

其中, vertex 域存放某个顶点的数据信息;
link 域存放某个链表中第一个结点的地址。

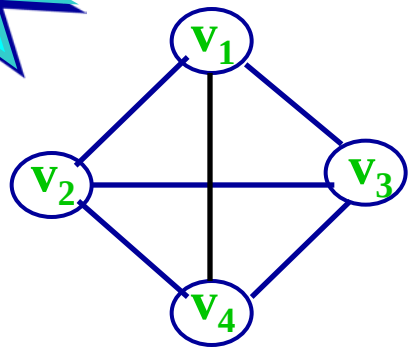
n个头结点之间为一数组结构.

2. 第i 个链表中的每一个链结点(称之为**边结点**)表示以第i个顶点为**出发点**的一条边;边结点的构造为

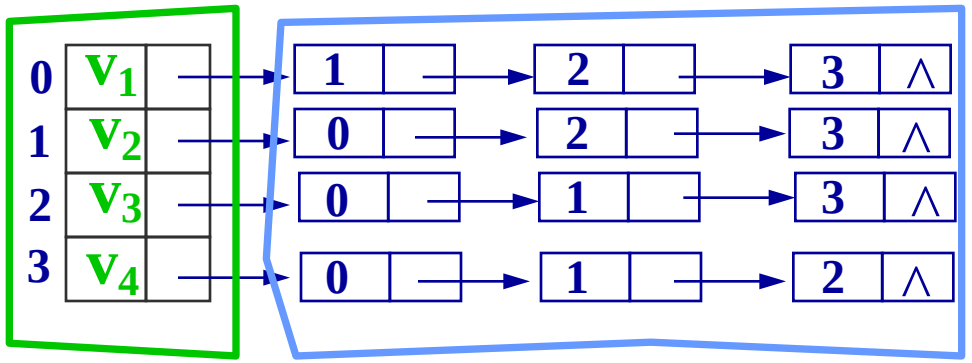
adjvex	weight	next

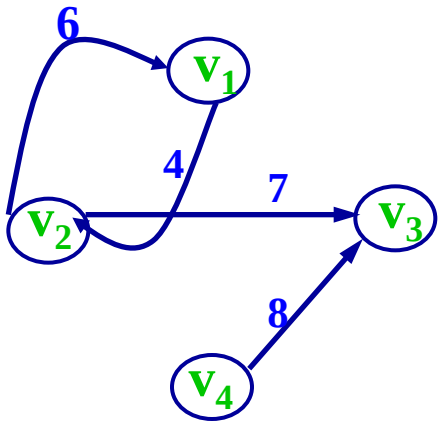
其中, next 域为指针域;
weight 域为权值域(若图不带权,则无此域);
adjvex 域存放以第i 个顶点为出发点的一条边的另一端点在头结点数组中的位置。

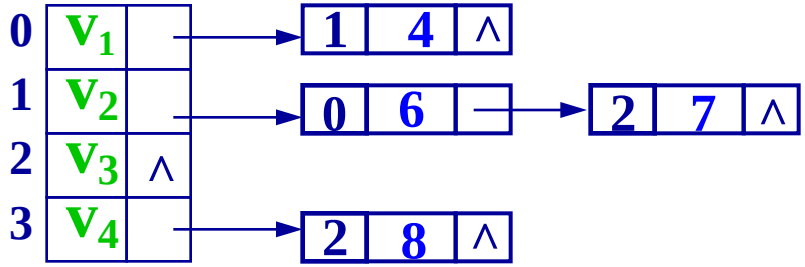

例

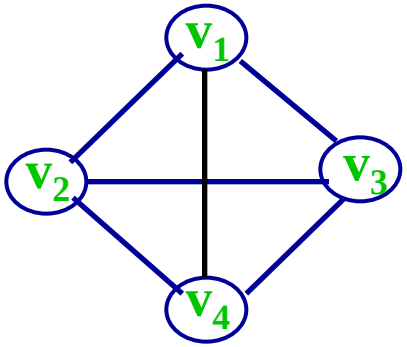

v1
v2
v3
v4

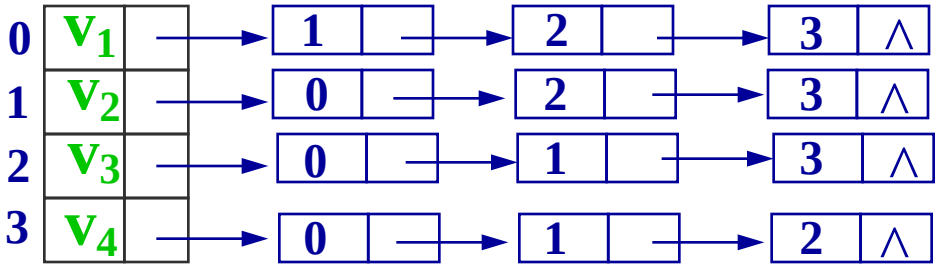
顶点结点　　边结点

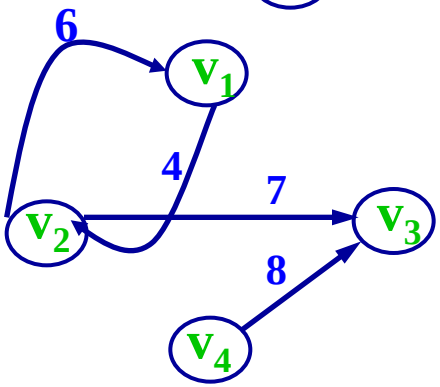

0 v1
1 v2
2 v3
3 v4
1 2 3 ∧
0 2 3 ∧
0 1 3 ∧
0 1 2 ∧

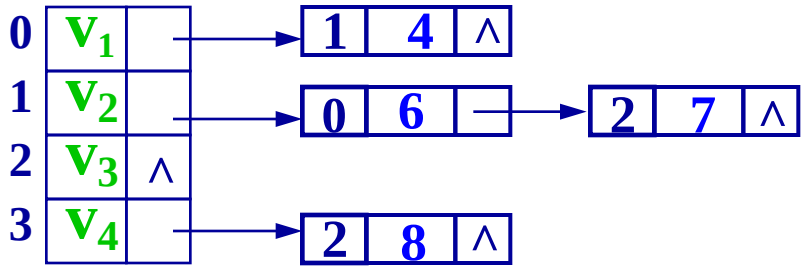

6
v1
4
7
v2
v3
8
v4


0 v1
1 v2
2 v3 ∧
3 v4
1 4 ∧
0 6
2 7 ∧
2 8 ∧









特点

(1). 无向图的第i 个链表中边结点的个数是第i 个顶点度数。

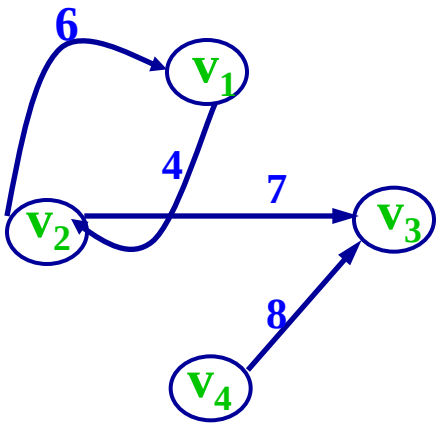
(2). 有向图[illegible]边结点[illegible]

(3)[illegible]，[illegible]图时，需要[illegible]顶点结点，2e个边结点；采用邻接表存储有向图时，需要n个顶点结点，e个边结点。

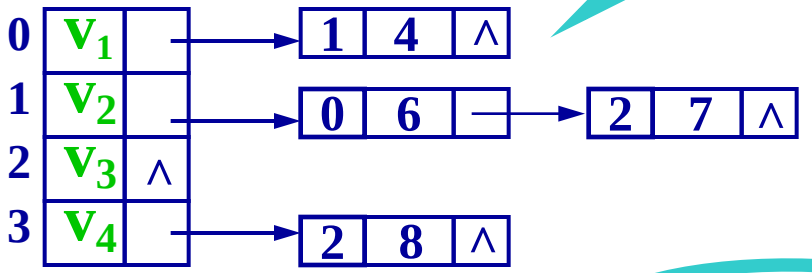
无向图的边结点个数一定为偶数！

边结点个数为奇数的图一定是有向图！

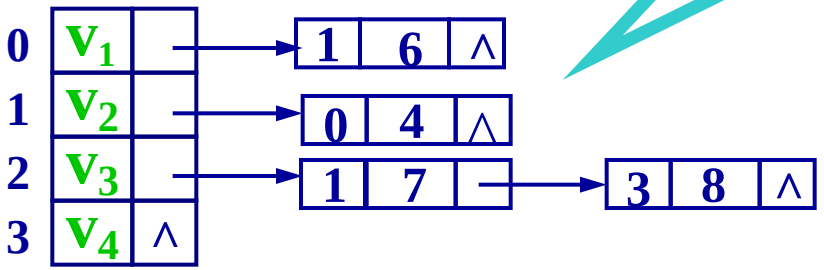
关于逆邻接表



邻接表



逆邻接表

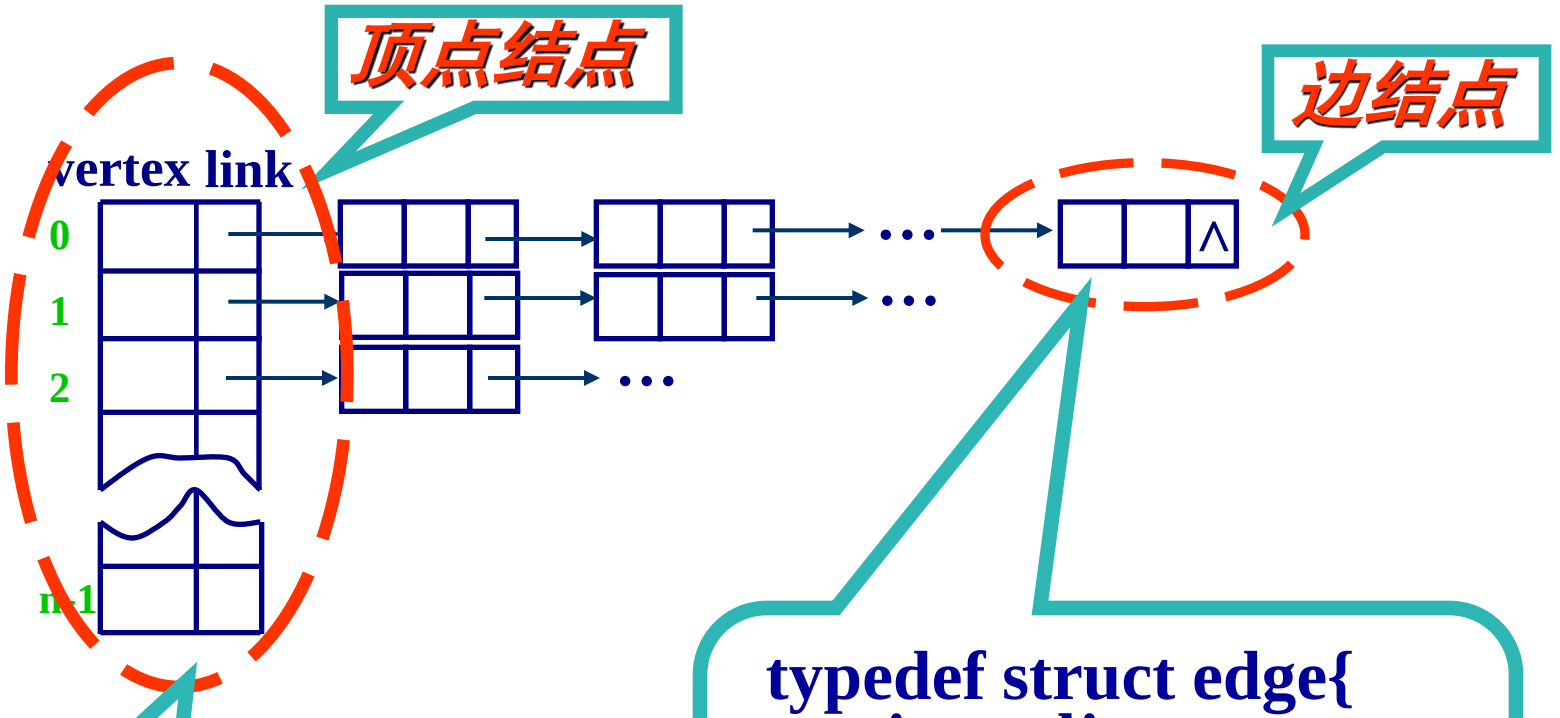


对于邻接表

第i个链表中的每一个链结点(称之为边结点)表示以第i个顶点为**出发点**的一条边；…

终止点

C语言描述



一个头结点

```
typedef struct ver{
    vertype  vertex;
    ELink  *link;
}VLink;
```

```
typedef struct edge{
    int  adjvex;
    int  weight;
    struct edge  *next;
}ELink;
```

VLink G[n];

```
#define MAXV    <最大顶点个数>

typedef struct  edge {
        int   adjvex;
        int   weight;
        struct edge   *next;
} ELink;

typedef struct ver {
        vertype   vertex;
        ELink   *link;
} VLink;

VLink  G[MAXV];
```

定义边结点类型

定义顶点结点类型

三. 有向图的十字链表存储方法

（略）

四. 无向图的多重邻接表存储方法

（略）

8.3 图的遍历

以无向图为例

从图中某个指定的顶点出发，按照某一原则对图中所有顶点都访问一次，得到一个由图中所有顶点组成的序列，这一过程称为图的遍历。

利用图的遍历

- 确定图中满足条件的顶点；
- 求解图的连通性问题，如求分量；
- 判断图中是否存在回路；
- …

一. 深度优先遍历

完成一个连通分量的遍历

类似于二叉树的前序遍历

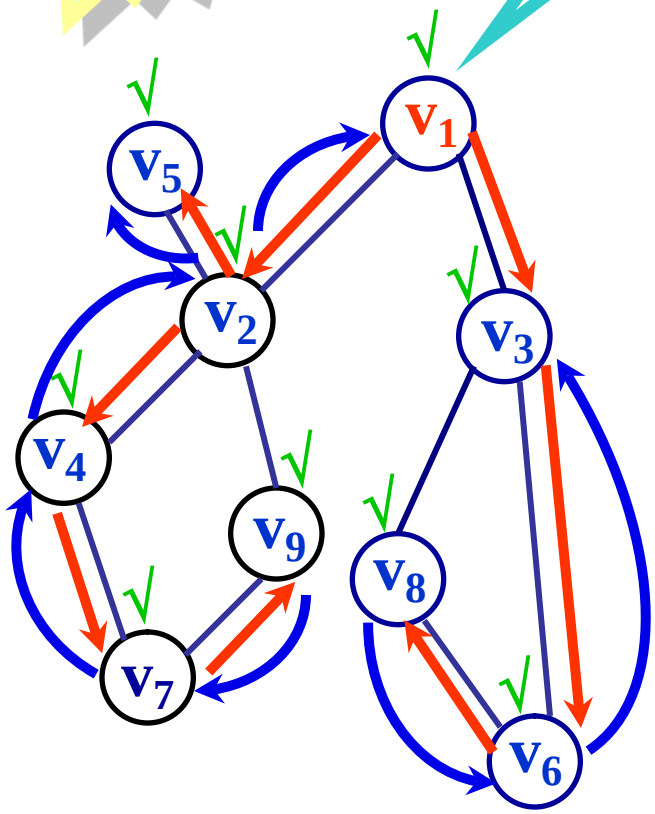
原则 从图中某个指定的顶点v出发，先访问顶点v，然后从顶点v未被访问过的一个邻接点出发，继续进行深度优先遍历，直到图中与v相通的所有顶点都被访问；若此时图中还有未被访问过的顶点，则从另一个未被访问过的顶点出发重复上述过程，直到遍历全图。

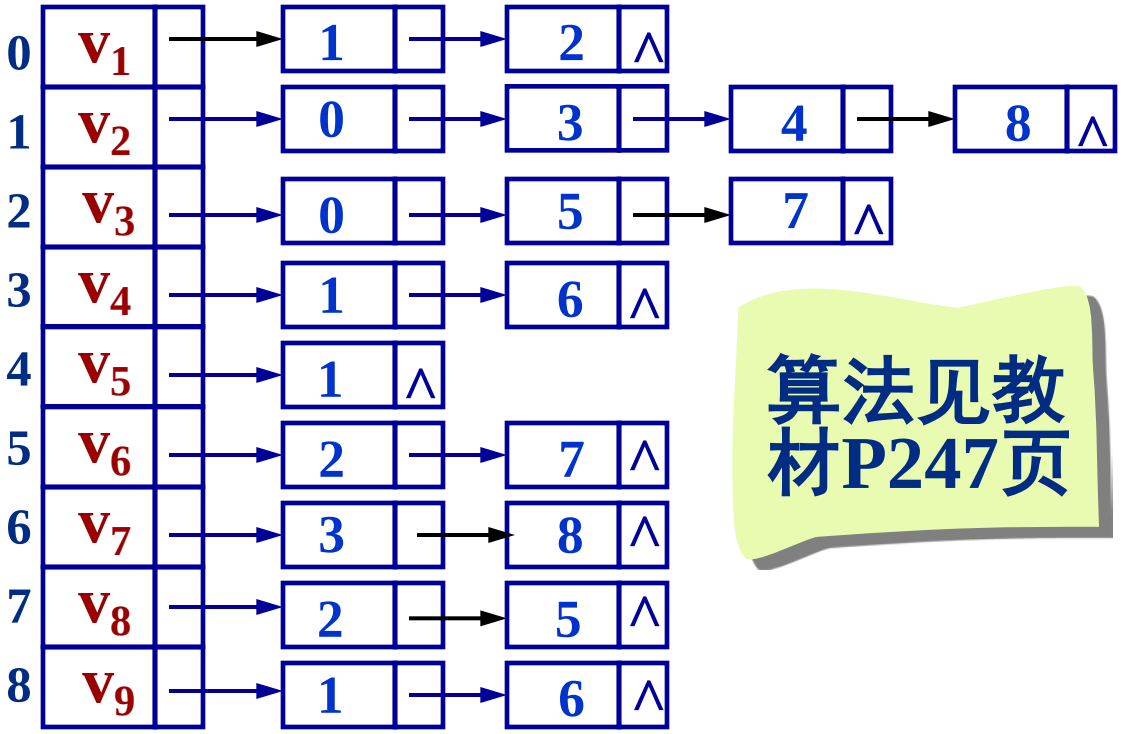
递归过程

例

出发点

关于遍历序列的惟一性





算法见教材P247页

v_1 v_2 v_4 v_7 v_9 v_5 v_3 v_6 v_8

遍历序列

算法分析

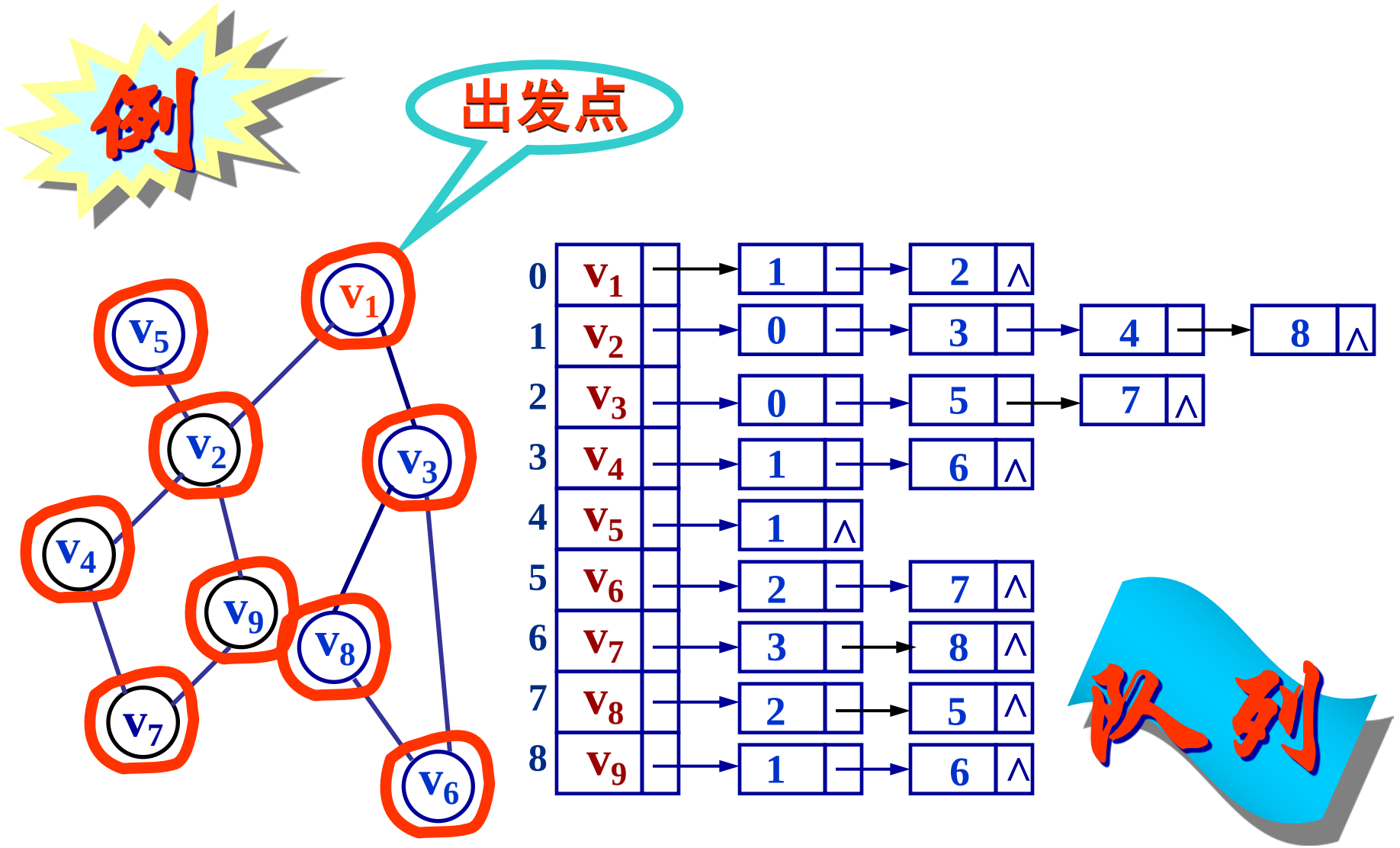
如果图中具有n个顶点、e条边，则

- 若采用邻接表存储该图，由于邻接表中有2e个或e个边结点，因而扫描边结点的时间为O(e)；而所有顶点都递归访问一次，所以，算法的时间复杂度为O(n+e)。
- 若采用邻接矩阵存储该图，则查找每一个顶点所依附的所有边的时间复杂度为O(n)，因而算法的时间复杂度为$O(n^2)$。

二. 广度优先搜索

原则 从图中某个指定的顶点v出发，先访问顶点v，然后依次访问顶点v的各个未被访问过的邻接点，然后又从这些邻接点出发，按照同样的规则访问它们的那些未被访问过的邻接点，如此下去，直到图中与v 相通的所有顶点都被访问；若此时图中还有未被访问过的顶点，则从另一个未被访问过的顶点出发重复上述过程，直到遍历全图。

完成一个连通分量的遍历


例
出发点
v1
v5
v2
v3
v4
v9
v8
v7
v6
0 v1 1 2 ∧
1 v2 0 3 4 8 ∧
2 v3 0 5 7 ∧
3 v4 1 6 ∧
4 v5 1 ∧
5 v6 2 7 ∧
6 v7 3 8 ∧
7 v8 2 5 ∧
8 v9 1 6 ∧
队列
V1 V2 V3 V4 V5 V9 V6 V8 V7
遍历序列

如何确定顶点不被重复访问

为了标记某一时刻图中哪些顶点是否被访问, 定义一维数组**visited[0..n-1]**，有

$$visited[i] = \begin{cases} 1 & \text{表示对应的顶点已经被访问} \\ 0 & \text{表示对应的顶点还未被访问} \end{cases}$$

算法分析

算法见教材P250页

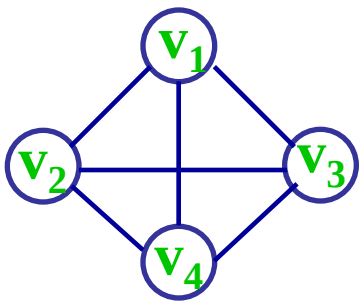
采用邻接表存储该图：**O(n+e)**。

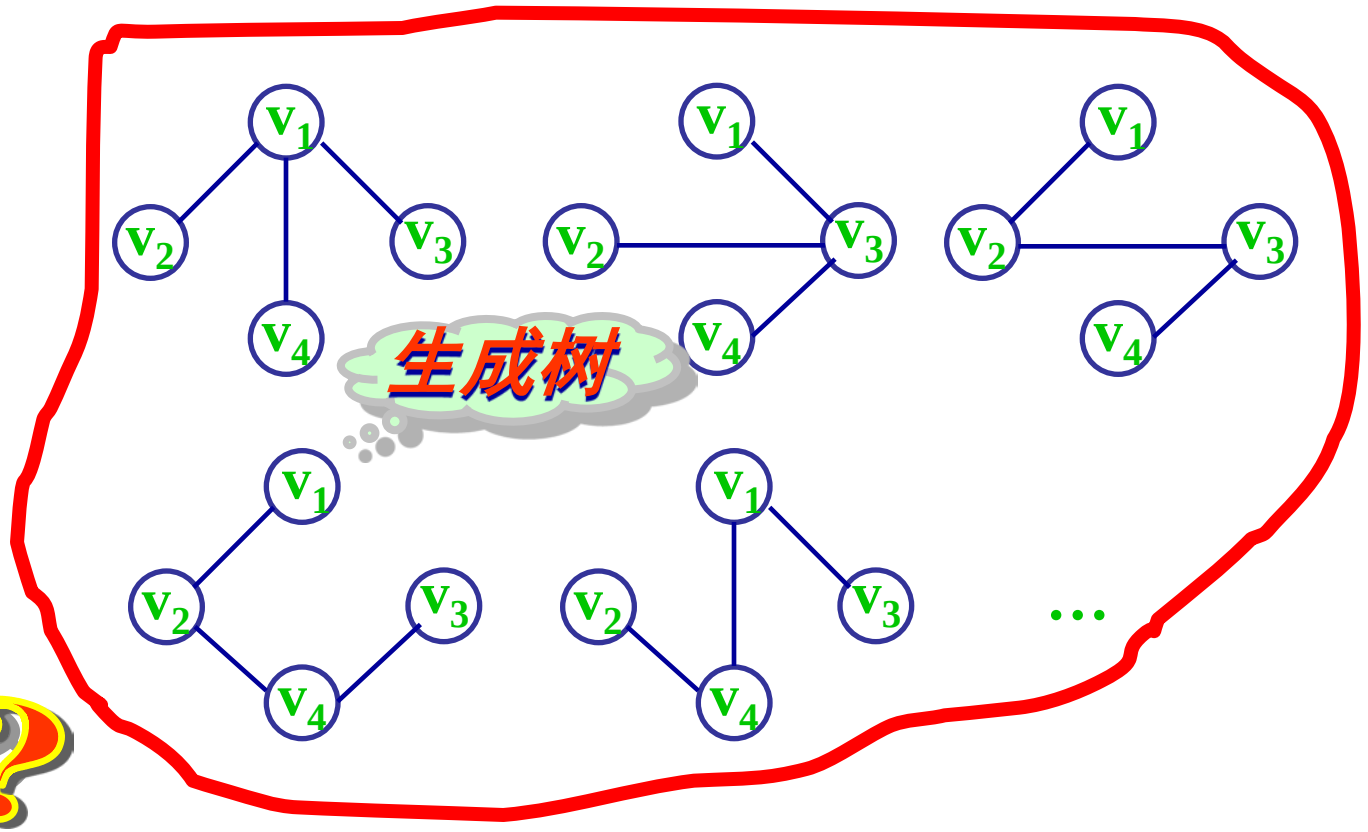
采用邻接矩阵存储该图：**$O(n^2)$**。

8.4 最小生成树

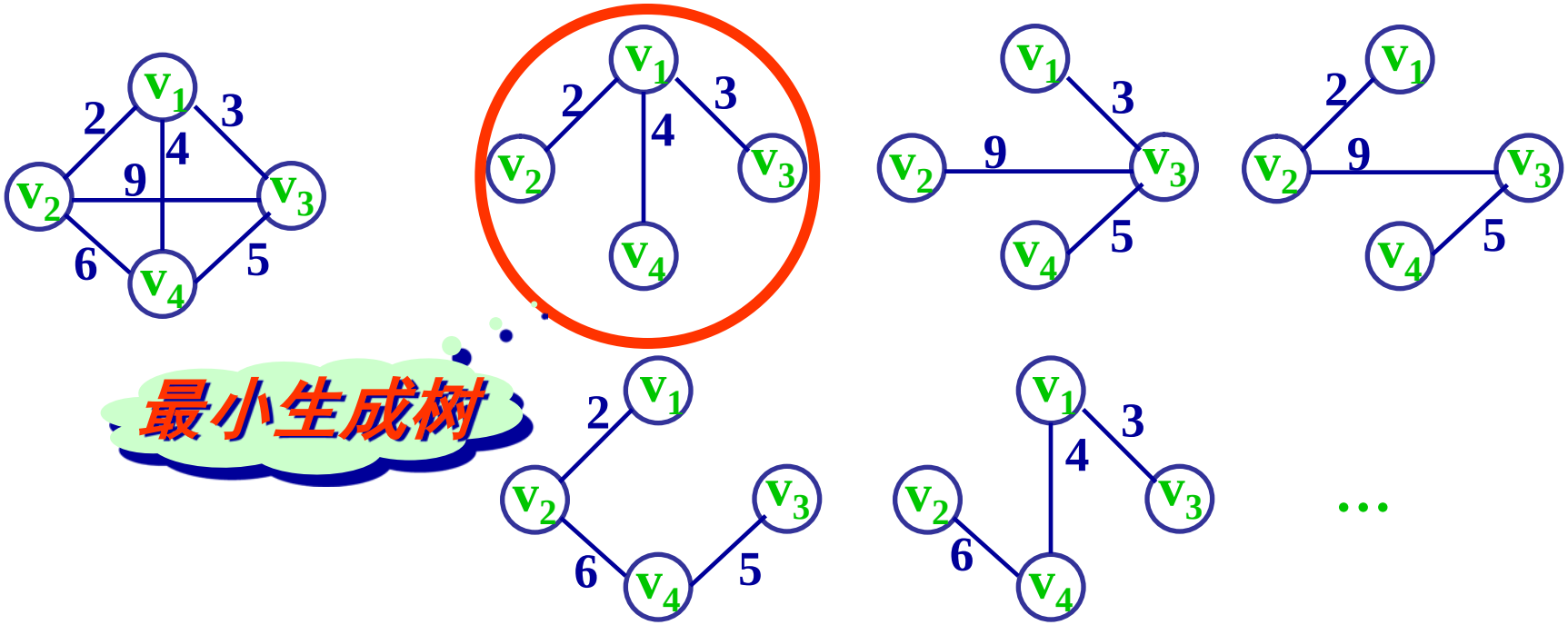
一. 什么是最小生成树

包含着连通图的全部n个顶点，仅包含其n-1条边的极小连通子图。





什么是生成树？



带权连通图中，总的权值之和最小的带权生成树为 ***最小生成树***。最小生成树也称最小代价生成树，或最小花费生成树。

构造最小生成树的原则

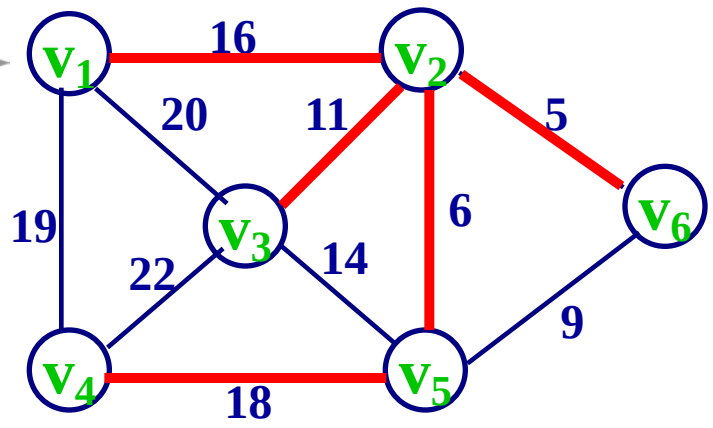
- 只能利用图中的边来构造最小生成树；
- 只能使用、且仅能使用图中的n-1条边来连接图中的n个顶点；
- 不能使用图中产生回路的边。

二. 求最小生成树

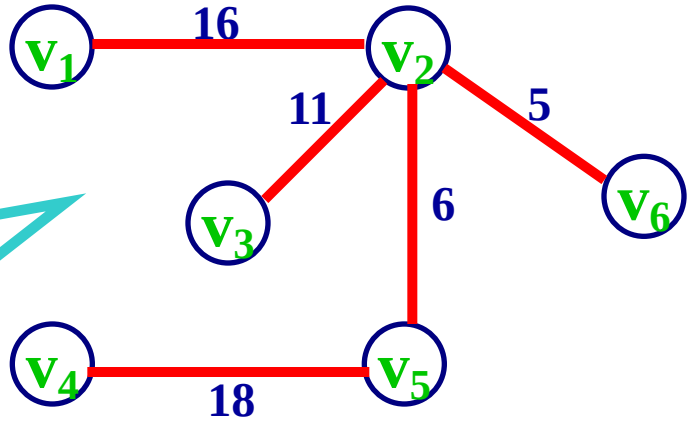
普里姆(Prim)算法:

设G=(V, GE)为具有n个顶点的带权连通图;
T=(U, TE)为正在生成的最小生成树,
初始时, TE=空, U={v_1}, $v_1 \in V$。

依次在G中选择一条一个顶点仅在V中, 另一个顶点在U中, 并且权值最小的边加入集合TE, 同时将该边仅在V 中的那个顶点加入集合U。 重复上述过程 n–1次, 使得U=V, 此时T为G的最小生成树。



最小生成树



最小生成树的权值 = 56

G:

$V = \{ v_1, v_2, v_3, v_4, v_5, v_6 \}$

$$GE = \{ (v_1, v_2)_{16}, (v_1, v_3)_{20}, (v_1, v_4)_{19}, (v_2, v_3)_{11}, (v_2, v_5)_{6}, (v_2, v_6)_{5}, (v_3, v_4)_{22}, (v_3, v_5)_{14}, (v_4, v_5)_{18}, (v_5, v_6)_{9} \}$$

T:

$U = \{ v_1, v_2, v_6, v_5, v_3, v_4 \}$

$TE = \{ (v_1, v_2)_{16} \ (v_2, v_6)_{5} \ (v_2, v_5)_{6} \ (v_2, v_3)_{11} \ (v_4, v_5)_{18} \}$

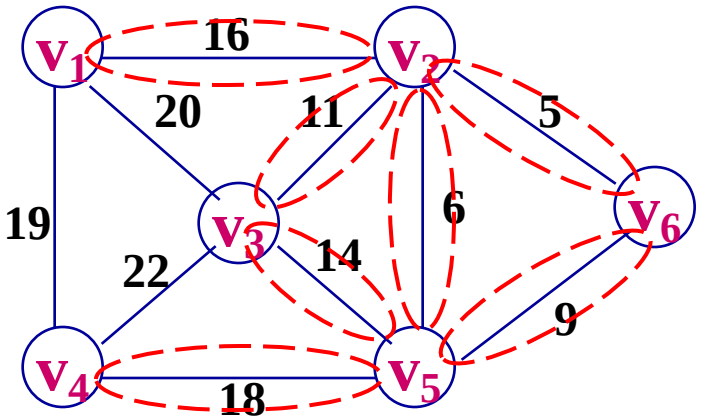
克鲁斯卡尔(Kruskal)方法

G=(V, GE)
T=(U, TE)
初始时, TE=空
U=V

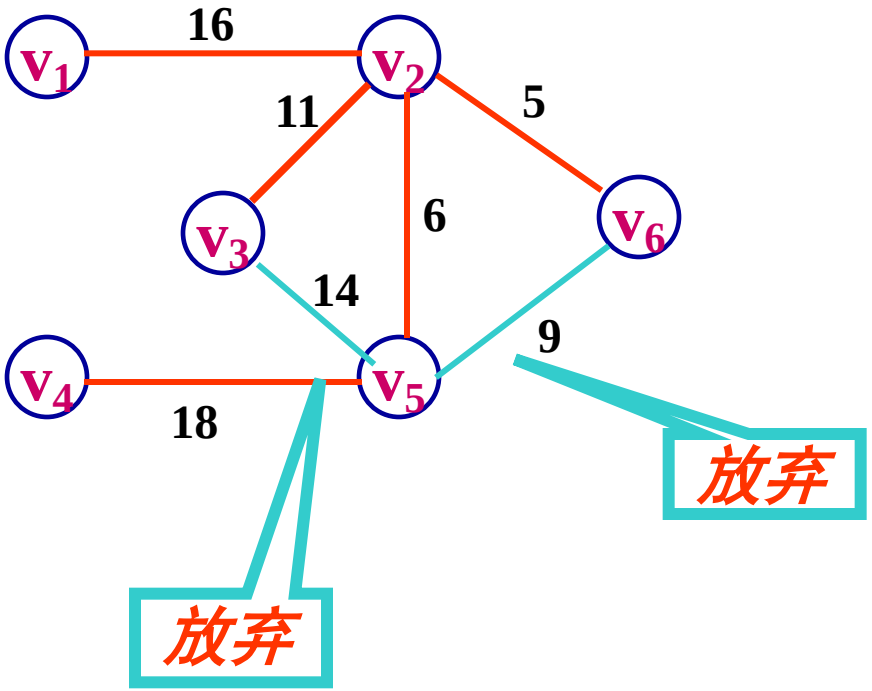
基本思想

从G中选择一条当前未选择过的、且边上的权值最小的边加入TE，若加入TE后使得T未产生回路，则本次选择有效；如使得T产生回路，则本次选择无效，放弃本次选择的边。重复上述选择过程直到TE中包含了n-1条边，此时的T为G的最小生成树。

G（连通图）


v1
v2
v3
v4
v5
v6
16
20
11
5
19
6
22
14
9
18

T（生成树）


v1
v2
v3
v4
v5
v6
16
11
5
6
14
9
18
放弃
放弃

例

8.5 最短路径问题

1. 单源点最短路径；√
2. 每对顶点之间的最短路径；
3. 求图中第1短、第2短、…的最短路径。
……

一. 路径长度的定义

1. 不带权的图：路径上所经过的边的数目。
2. 带权的图：路径上所经过的边上的权值之和。

二. 问题的提出

设出发顶点为v(通常称为源点)。

三. 解决问题所需要确定的数据结构

1. 图的存储：以1~n 分别代表n个顶点，采用邻接矩阵存储该图，有

$$A[i,j]=\begin{cases} w_{ij} & \text{当顶点}v_i\text{到顶点}v_j\text{有边，且权为}w_{ij} \\ \infty & \text{当顶点}v_i\text{到顶点}v_j\text{无边时} \\ 0 & \text{当}v_i=v_j\text{时} \end{cases}$$

2. 设置一个标志数组s[0:n-1]记录源点v 到图中哪些顶点的最短路径已经找到，有：

$$s[i] = \begin{cases} 1 & \text{表示源点到顶点i 的最短路径已经找到} \\ 0 & \text{表示源点到顶点i 的最短路径尚未找到} \end{cases}$$

初始时， s[v]=1,　s[i]=0　　i =0,1, 2, …, n-1　　i ≠v

3. 设置数组dist[0:n-1] 分别记录源点v 到图中各顶点的最短路径的路径长度，　其中，dist[i]记录源点到顶点i 的最短路径的长度；初始时，dist数组的值为邻接矩阵第v 行的n个元素值。

4. 设置数组path[0:n-1] 分别记录源点v 到图中各顶点的最短路径所经过的顶点序列，　其中，path[i]记录源点到顶点i 的路径，初始时，path[i]={ v },　　i =0,1, 2, 3, …, n-1

四. 算法(用自然语言表达)

```
for i←1 to n do
    dist[i]←A[v, i]
    s[i]←0
    path[i]←{ v }
end
s[v]←1
```

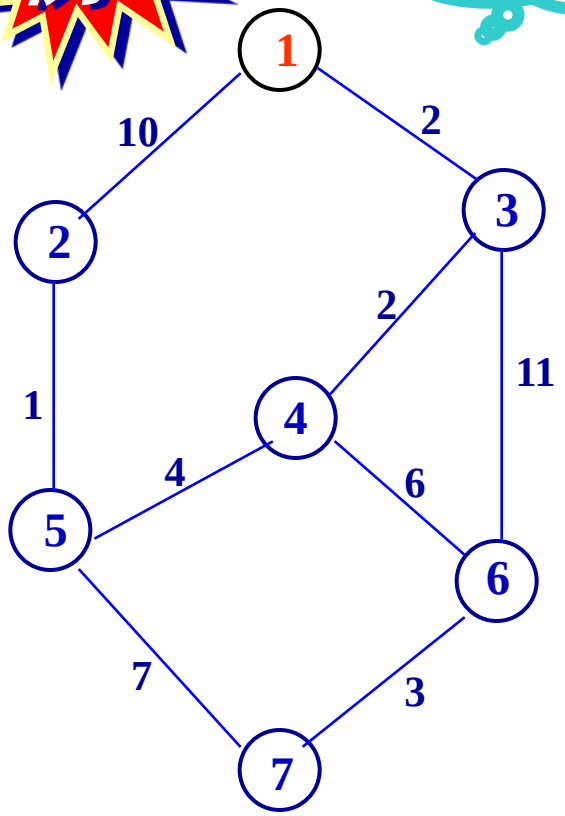
1. 确定dist、s、path 三个数组的初值。
2. 利用s数组与dist 数组在那些尚未找到最短路径的顶点中确定一个与源点最近的顶点u, 并置s[u]为1，同时将顶点u加入path[u].
3. 根据顶点u修改源点到所有尚未找到最短路径的顶点的路径长度。 即

   1）将源点v到顶点u的(最短)路径长度分别加到源点v通过顶点u可以直接到达、且尚未找到最短路径的那些顶点的路径长度上;若加后的长度小于原来v 到某顶点r的路径长度，则用加后的长度替换原来的长度，否则不替换。

   2）若替换，将源点v 到顶点u 的路径(最短路径)上经过的所有顶点替换path[r].
4. 重复上述过程的第2至第3步n–1 次。

例

源点：v =1



$$\begin{pmatrix} 0 & 10 & 2 & \infty & \infty & \infty & \infty \\ 10 & 0 & \infty & \infty & 1 & \infty & \infty \\ 2 & \infty & 0 & 2 & \infty & 11 & \infty \\ \infty & \infty & 2 & 0 & 4 & 6 & \infty \\ \infty & 1 & \infty & 4 & 0 & \infty & 7 \\ \infty & \infty & 11 & 6 & \infty & 0 & 3 \\ \infty & \infty & \infty & \infty & 7 & 3 & 0 \end{pmatrix}$$

dist数组

0	10	2	∞	∞	∞	∞

s数组

1	0	0	0	0	0	0

path数组

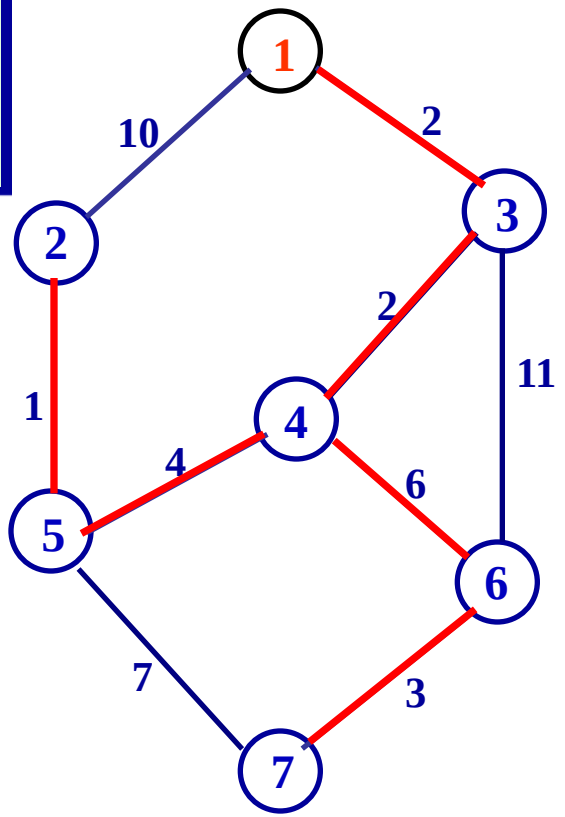
1	1	1	1	1	1	1

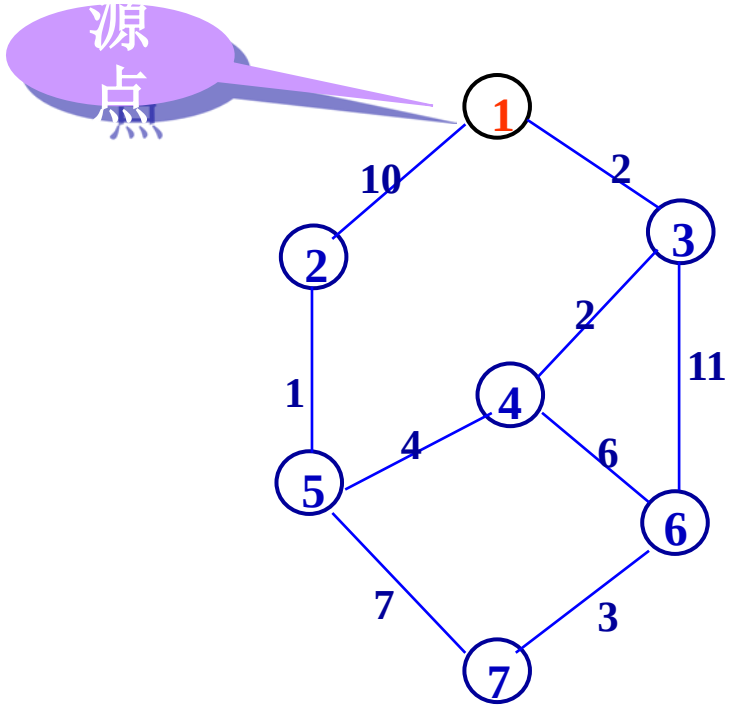
顶点	1	2	3	4	5	6	7
dist	0	10	2	∞	∞	∞	∞
	0	10	2	4	∞	13	∞
	0	10	2	4	8	10	∞
	0	9	2	4	8	10	15
	0	9	2	4	8	10	15
	0	9	2	4	8	10	13
	0	9	2	4	8	10	13
s	1	1	1	1	1	1	1
path	1	1 3 4 5 2	1 3	1 3 4	1 3 4 5	1 3 4 6	1 3 4 6 7

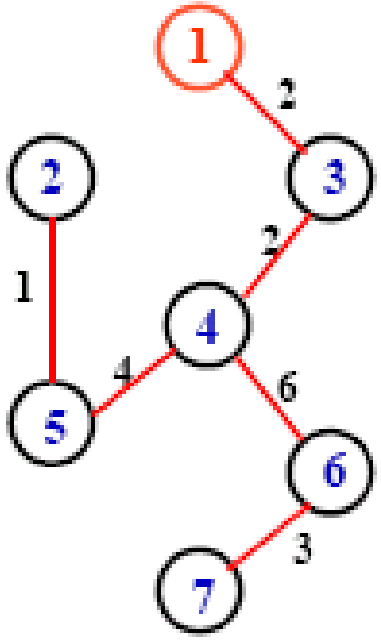
路径长度

路径

源点：v =1



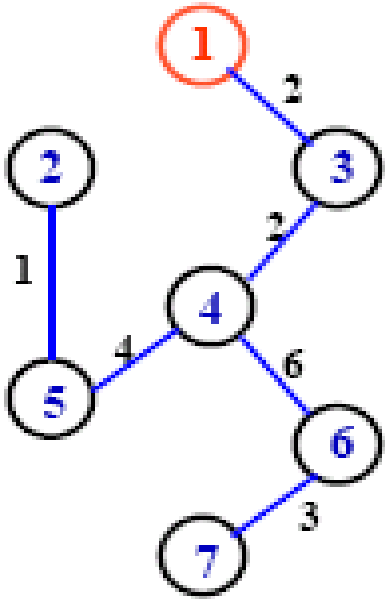




最短（路径）生成树

相同

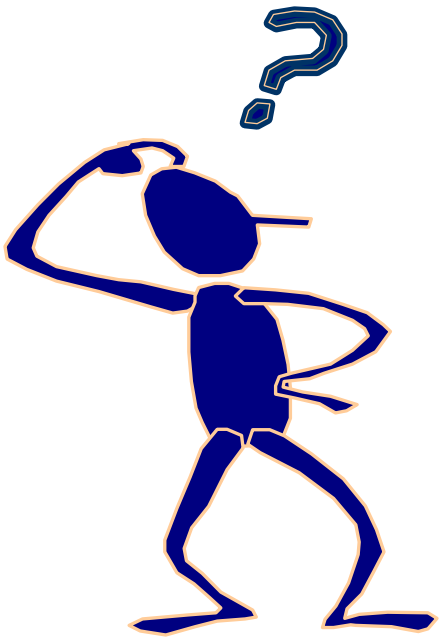
最小代价生成树



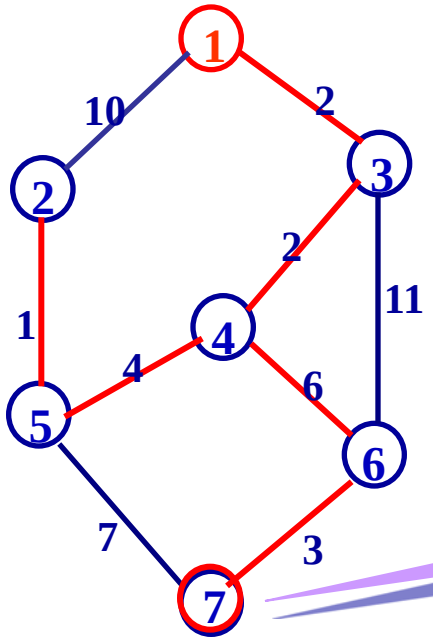
思考

对于给定的带权连通无向图，从某源点到图中各顶点的最短路径构成的生成树是否是该图的最小生成树？

为什么？

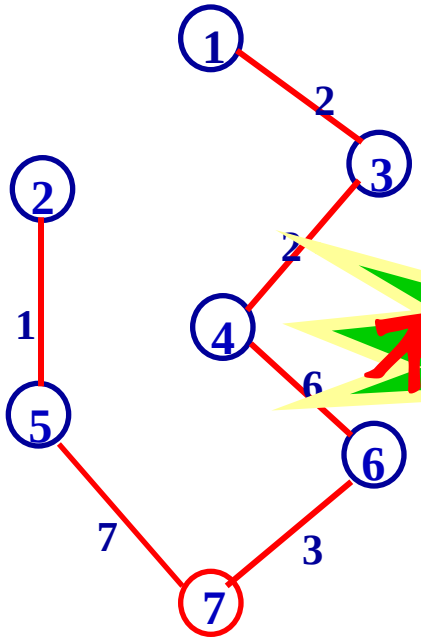


与源点有关！

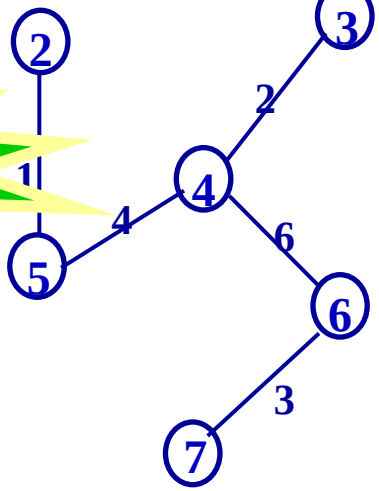


与“源点”无关！

最短（路径）生成树

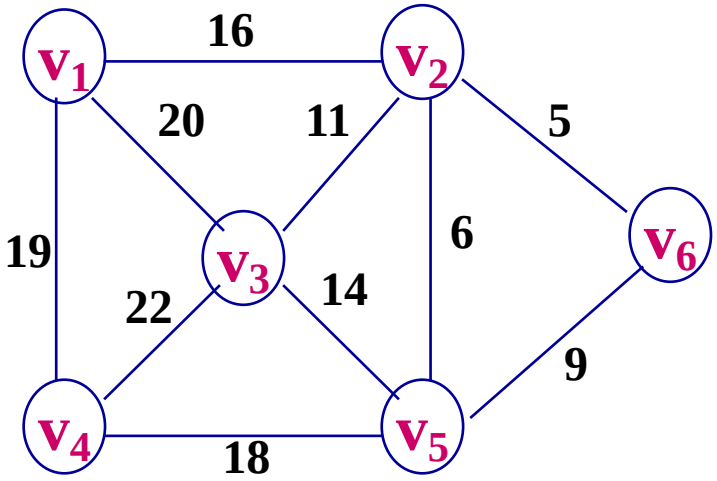


不相同！

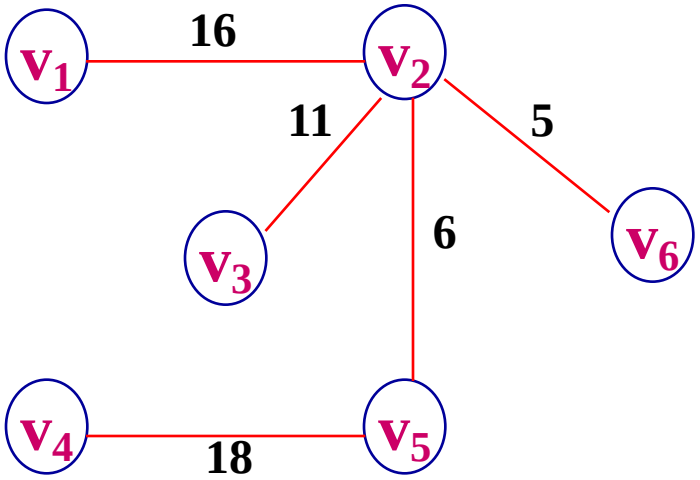


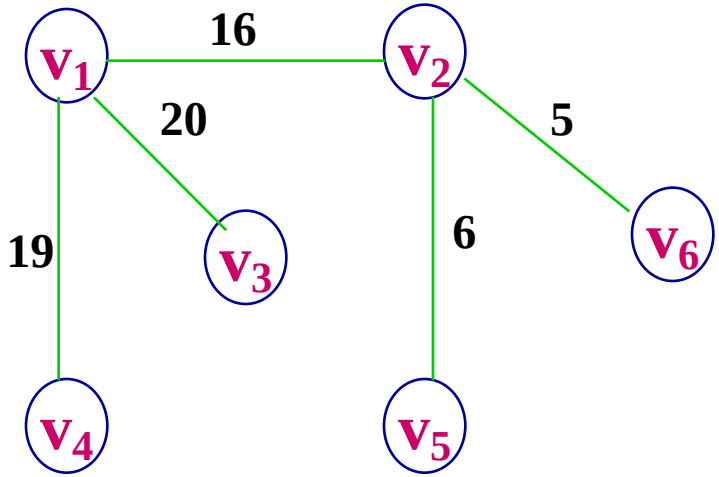
最小代价生成树

例



结论...





最小生成树

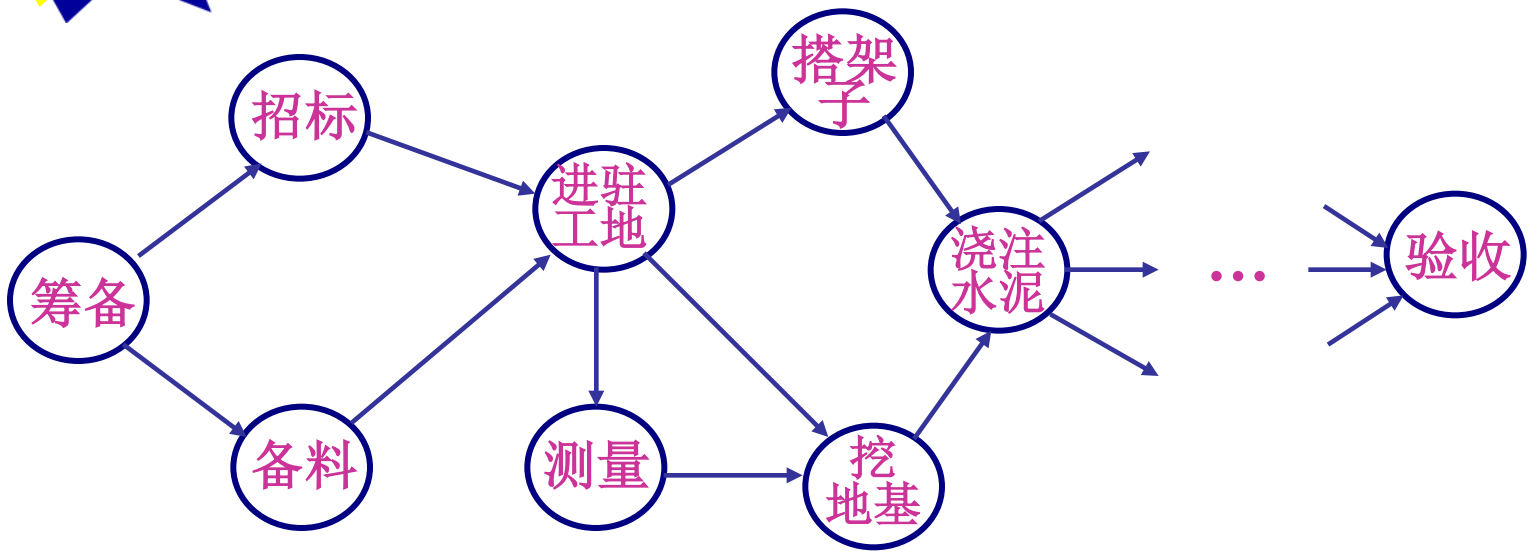
源点为V_1的最短路径生成树

8.6 AOV网与拓扑排序

一. 什么是AOV网

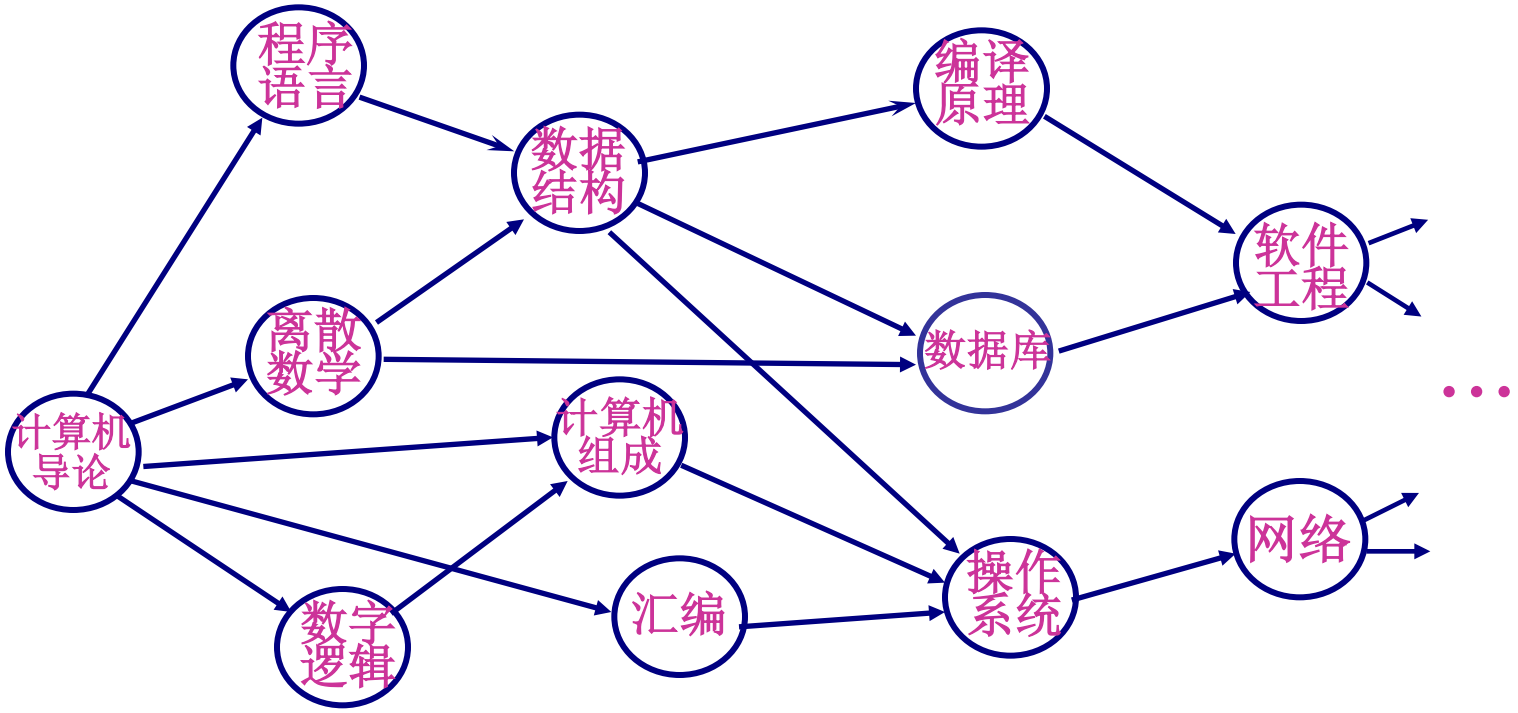
例1

一个建筑工程的施工流程



例2

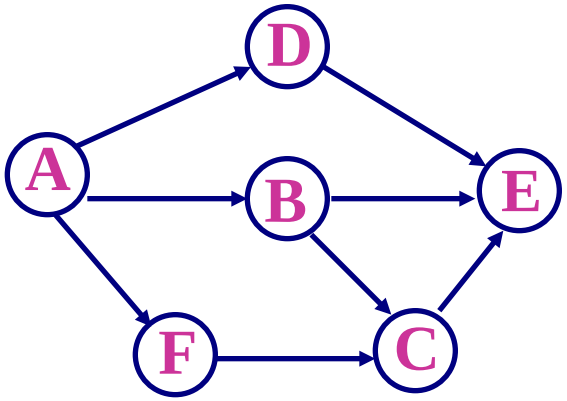
计算机专业专业课程教学流程安排



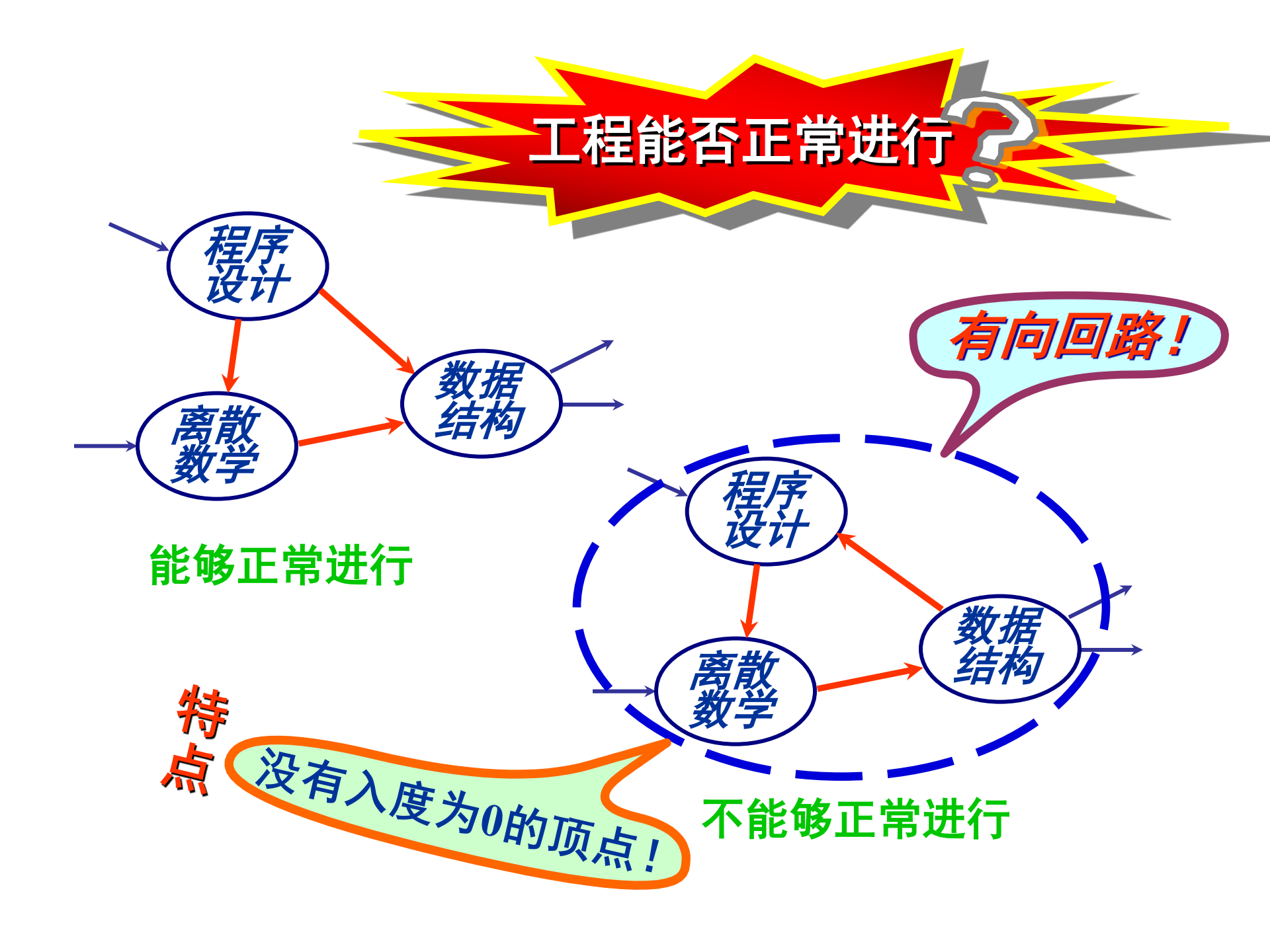
AOV网的定义

以顶点表示活动，以有向边表示活动之间的优先关系的有向图称为顶点表示活动的网(Activity On Vertex Network)，简称AOV网。

在AOV网中，若顶点i到顶点j 之间有有向路径，则称顶点i 为顶点j 的前驱，顶点j 为顶点i的后继； 若顶点i 到顶点j 之间为一条有向边，则称顶点i 为顶点j 的直接前驱，顶点j 为顶点i 的直接后继。



一个AOV网


工程能否正常进行？
程序设计
离散数学
数据结构
能够正常进行
有向回路！
程序设计
离散数学
数据结构
不能够正常进行
特点
没有入度为0的顶点！

检测工程能否正常进行，首先要判断AOV网中是否存在回路，达到该目的最有效的方法是对有向图构造其顶点的拓扑序列，即对AOV进行**拓扑排序**。

离散数学

由某个集合上的一个偏序得到该集合上的一个全序的操作称为**拓扑排序**

二. 拓扑排序

检查AOV网是否存在回路的方法是对AOV网进行***拓扑排序***，即构造一个序列，使得该序列满足条件：

1. 若在AOV网中，顶点i 优先于顶点j , 则在该序列中也是顶点i 优先于顶点j 。

2. 若在AOV网中， 顶点i 与顶点j之间不存在优先关系, 则在该序列中建立它们的优先关系，即顶点i 优先于顶点j，或顶点j 优先于顶点i 。

3. 若能够构造出拓扑序列，则拓扑序列包含AOV网的全部顶点。说明AOV网中没有回路。

拓扑序列的特点

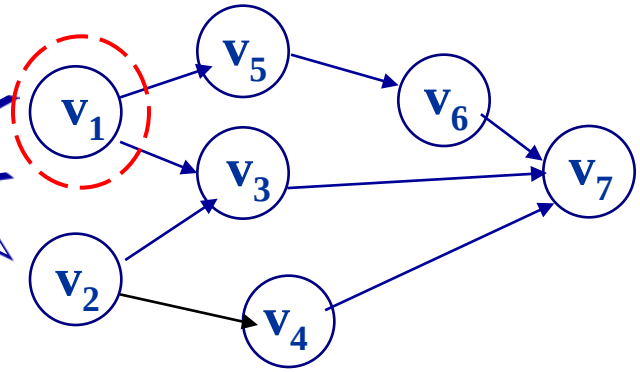
三. 拓扑排序方法

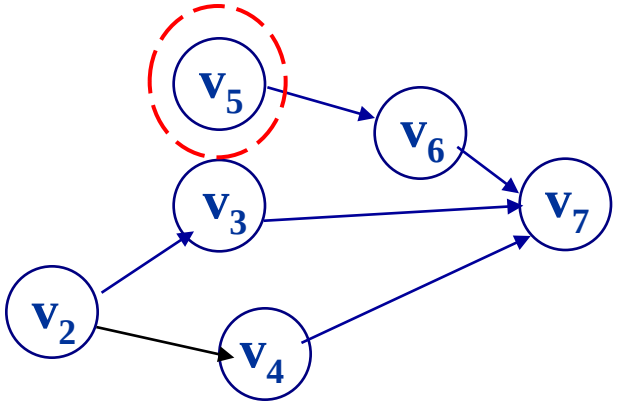
1. 从AOV网中任选一个没有前驱(入度为0)的顶点；

2. 从AOV网中去掉该顶点以及以该顶点为出发点的所有边；

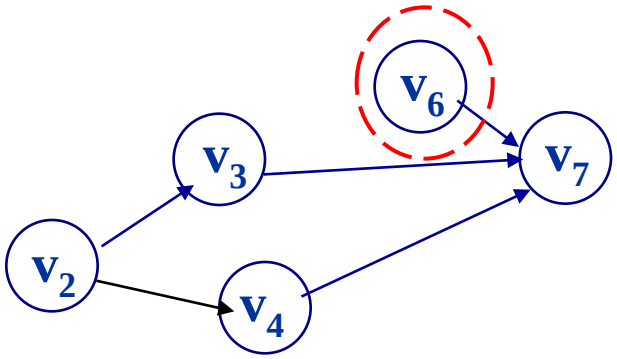
3. 重复上述过程，直到网中的所有顶点都被去掉，或者网中还有顶点，但不存在入度为0 的顶点。

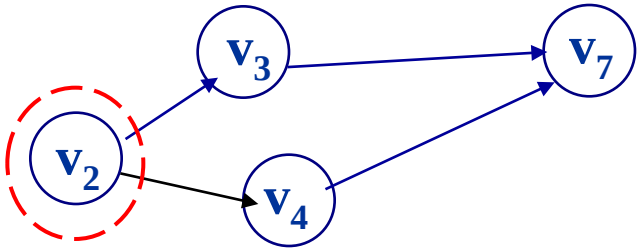
前者说明AOV网中无回路，
后者说明AOV网中存在回路。

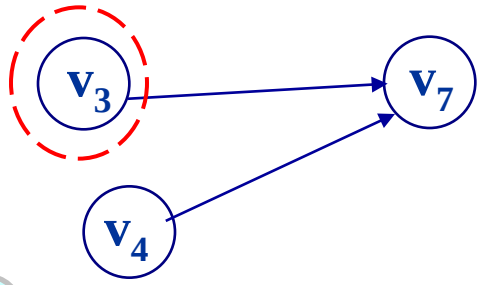


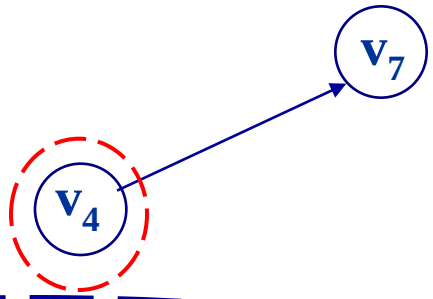


拓扑序列不是唯一的





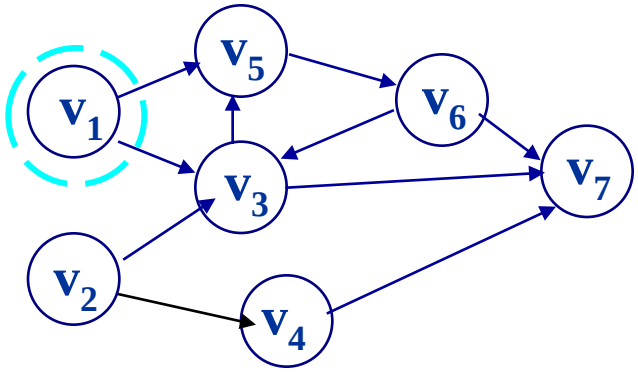


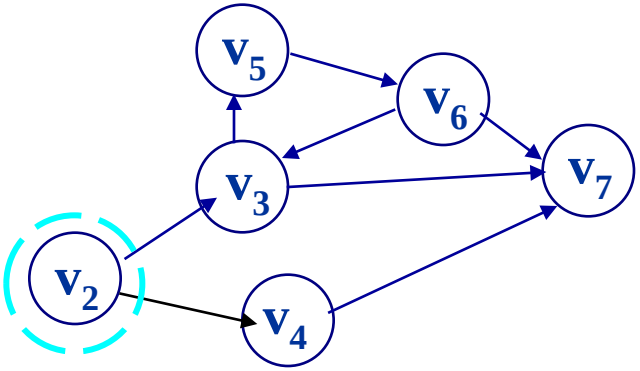


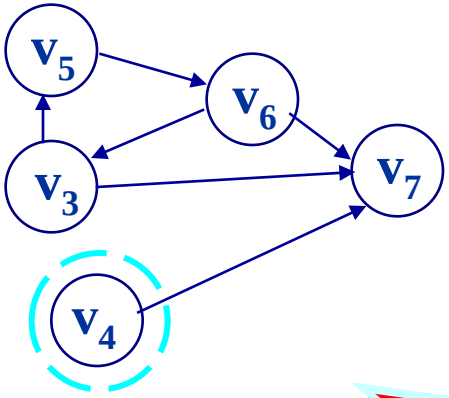
（空）

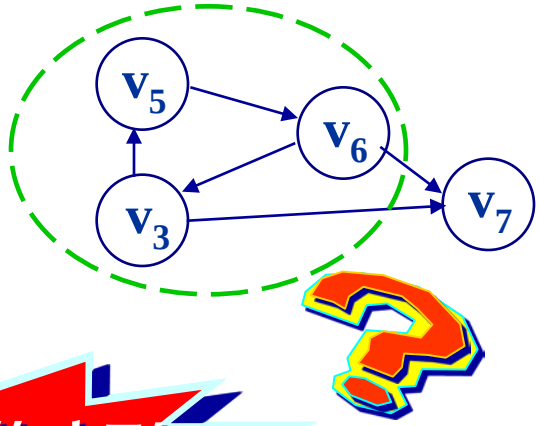
拓扑序列

$v_1\ v_5\ v_6\ v_2\ v_3\ v_4\ v_7$









$v_1\ v_2\ v_4$

不存在拓扑序列
网中有回路！

用自然语言描述的算法

采用邻接表存储

1. 首先建立一个入度为0的顶点堆栈，将网中所有入度为0的顶点分别进栈。

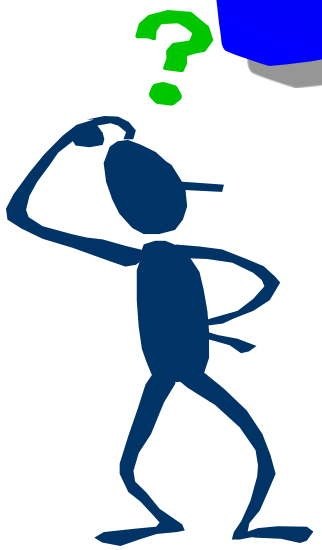
2. 当入度为0的顶点栈不空时，反复执行以下动作：
   - 从顶点栈中退出一个顶点，并输出它；
   - 从AOV网中删去该顶点以及以它发出的所有边，并同时分别将这些边的终点的入度减1；
   - 若此时边的终点的入度为0，则将该顶点进栈；

3. 若输出的顶点个数少于AOV网中的顶点个数，则报告网中存在有向回路，否则，说明该网中不存在回路。

算法详见教材P266—267

思考

除了进行拓扑排序，还可以采用什么方法判断一个有向图是否存在回路？

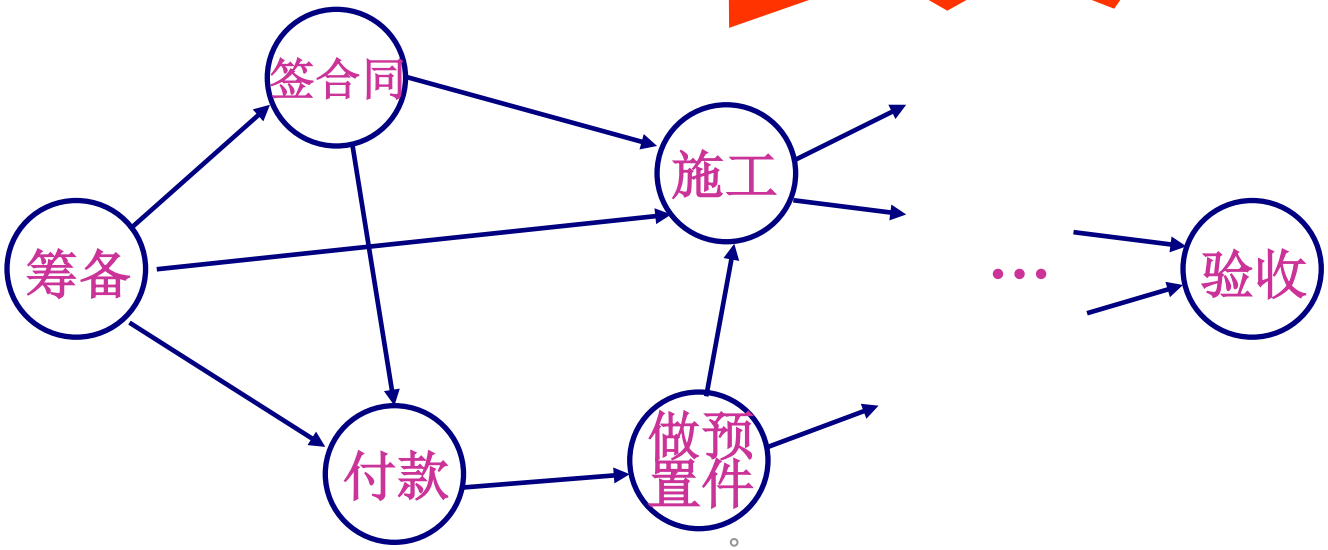


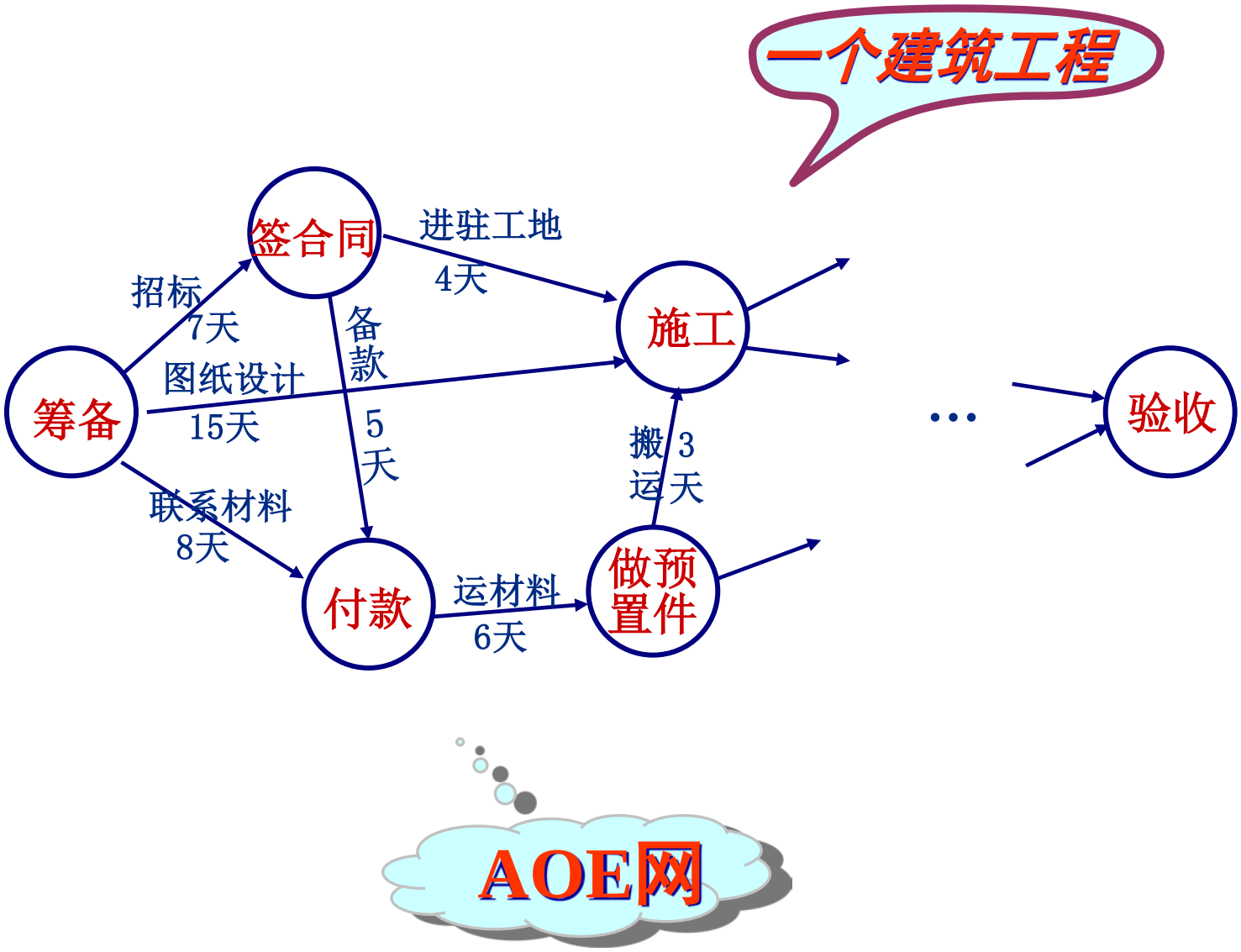
进行深度优先遍历。若从某个顶点v出发，遍历结束前出现了从顶点u到顶点v的回边，则可以断定图中包含顶点v到顶点u的回路。

8.7 AOE网与关键路径

关心

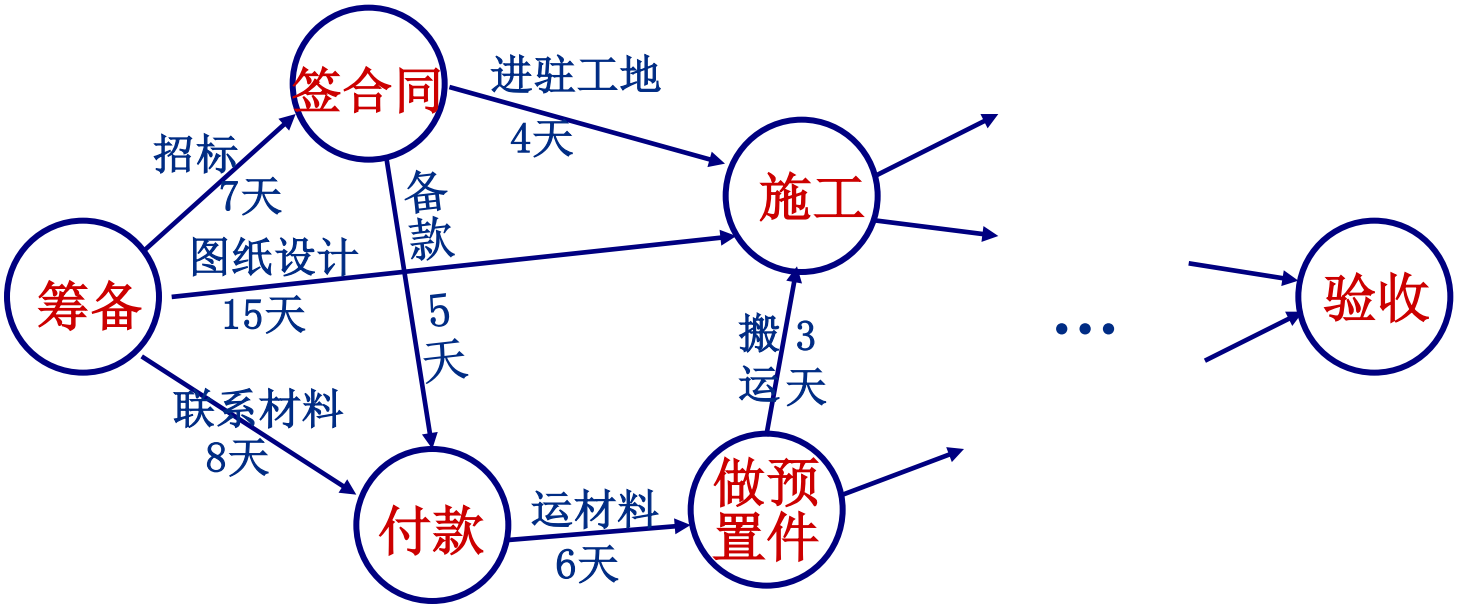
1. 每个活动持续多少时间？
2. 完成整个工程至少需要多少时间？
3. 哪些活动是关键活动？



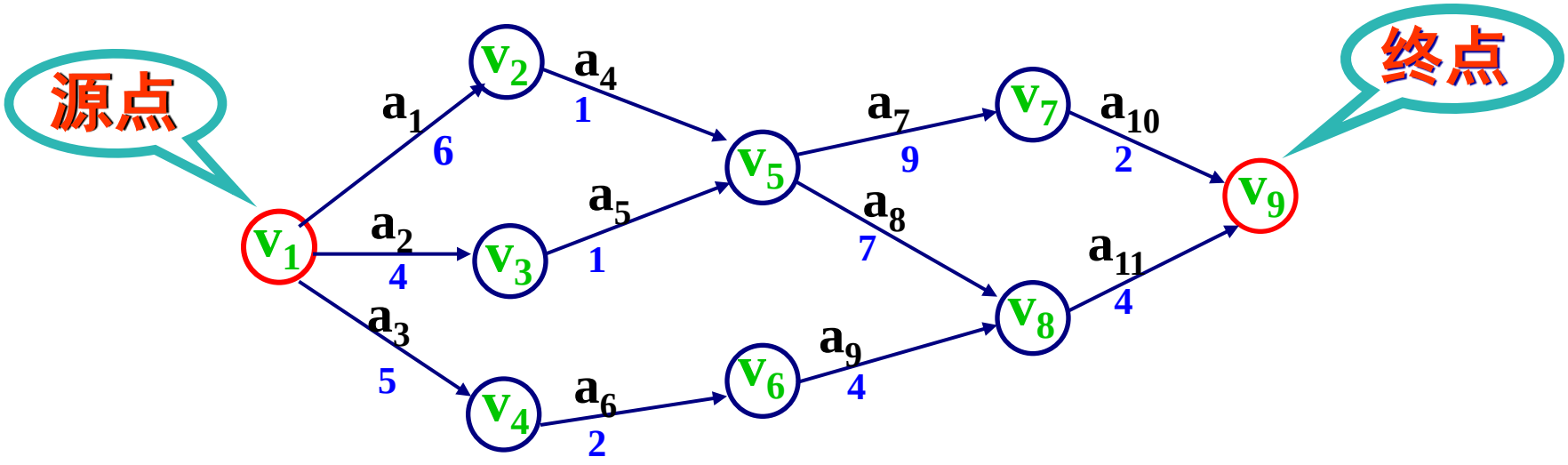

一个建筑工程
签合同
进驻工地
4天
招标
7天
备款
5天
施工
筹备
图纸设计
15天
…
验收
搬运
3天
联系材料
8天
付款
运材料
6天
做预置件
AOE网

一. AOE网的定义

AOE网为一个带权的有向无环图，其中，顶点表示事件，有向边表示活动，边上的权值表示活动持续的时间。



一个AOE网



AOE网

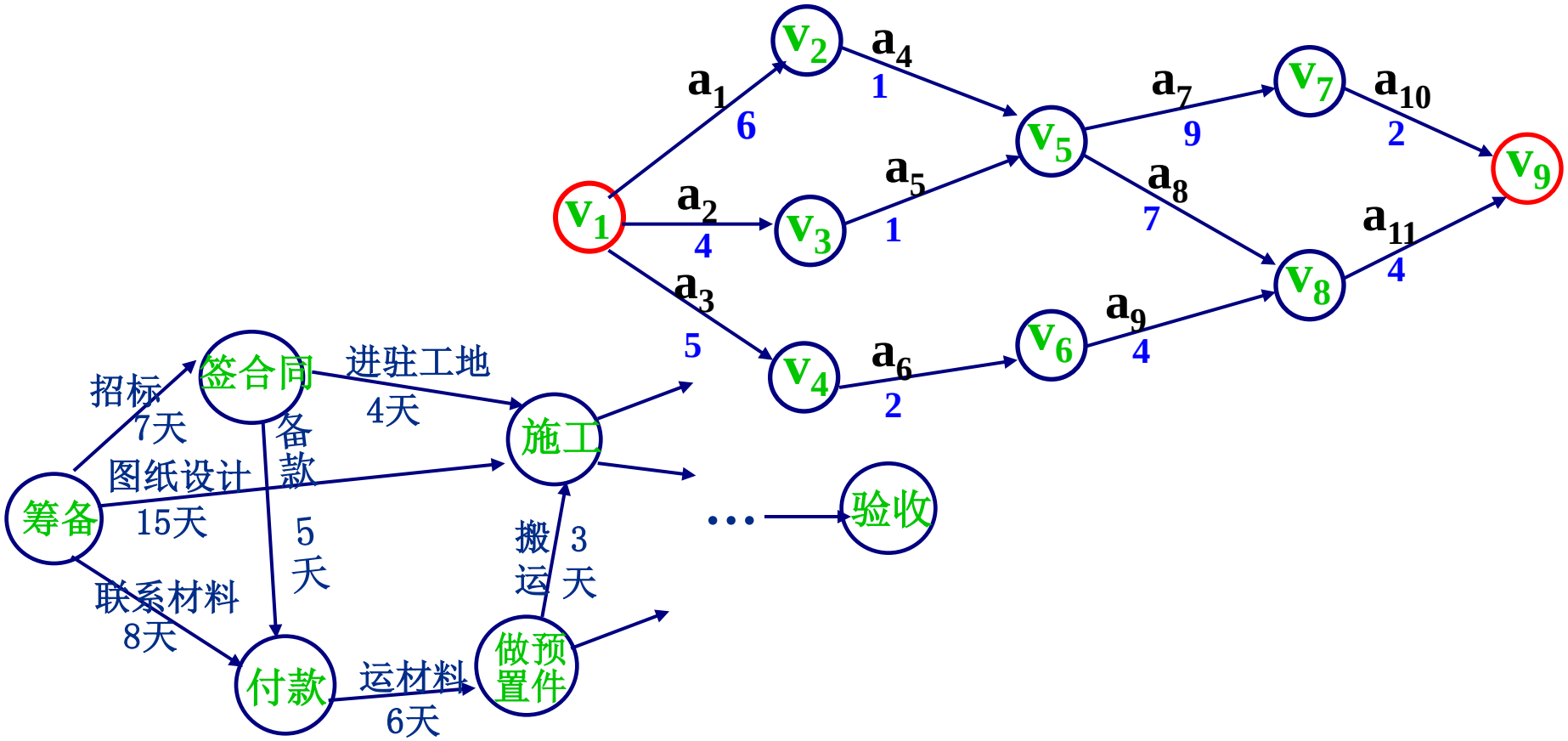
正常情况下，AOE网中只有一个入度为0的顶点，称之为**源点**；有一个出度为0 的顶点，称之为**终点**。

网中无回路

AOE网的特点

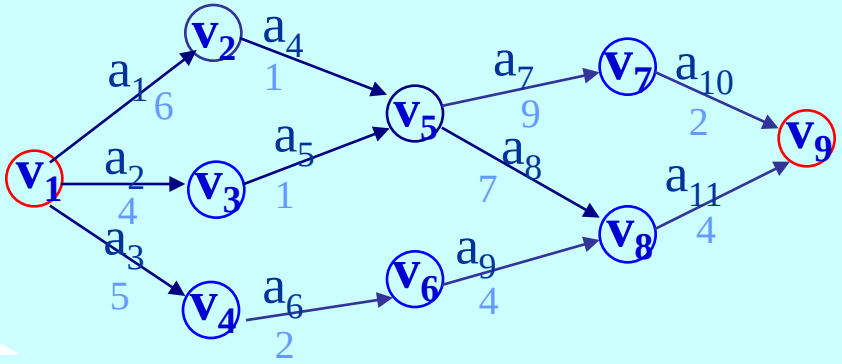
1. 只有在某个顶点所代表的事件发生以后，该顶点引发的活动才能开始。

2. 进入某事件的所有边代表的活动都已完成，该顶点代表的事件才能发生。



二. AOE网的存储方法

采用邻接矩阵存储方法。



三. 关键路径

1. 关键路径的定义

从源点到终点的路径中具有最大路径长度的路径为关键路径；关键路径上的活动称为关键活动。

2. 关键路径的特点

1) 关键路径的长度（路径上的边的权值之和）为完成整个工程所需要的最短时间。

2) 关键路径的长度变化（即任意关键活动的权值变化）将影响整个工程的进度，而其他非关键活动在一定范围内的变化不会影响工期。

求关键活动的思路

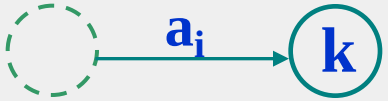
从源点到事件k的最长路径长度

e[i] — 活动a_i的最早开始时间
l[i] — 活动a_i的最迟开始时间
若l[i]–e[i]=0，则说明活动a_i为一个关键活动。

缓冲时间/松弛时间/时间余量

ee[k]— 事件k的最早发生时间

le[k] — 事件k的最迟发生时间



结 论

事件k的最早发生时间ee[k]
事件k的最迟发生时间le[k]

⇒ 活动a_i的最早开始时间e[i]
活动a_i的最迟开始时间l[i]

⇓

a_i为关键活动

求 e[i] = l[i]

四. 求关键路径

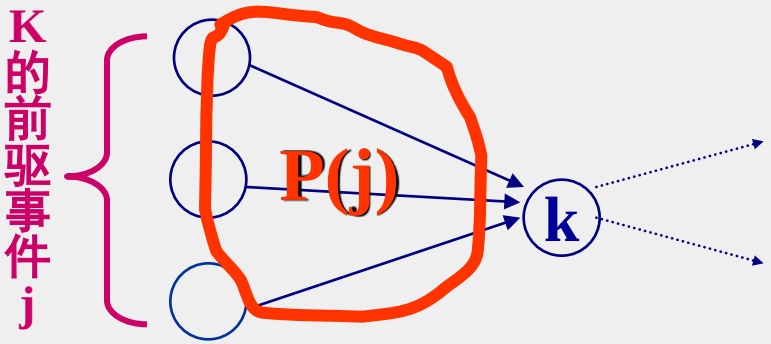
1. 计算事件k的最早发生时间ee[k]

所谓事件k的最早发生时间ee[k]是指从源点到顶点(事件)k的最大路径长度；该时间决定了由事件k出发的活动的最早开始时间.

计算方法:

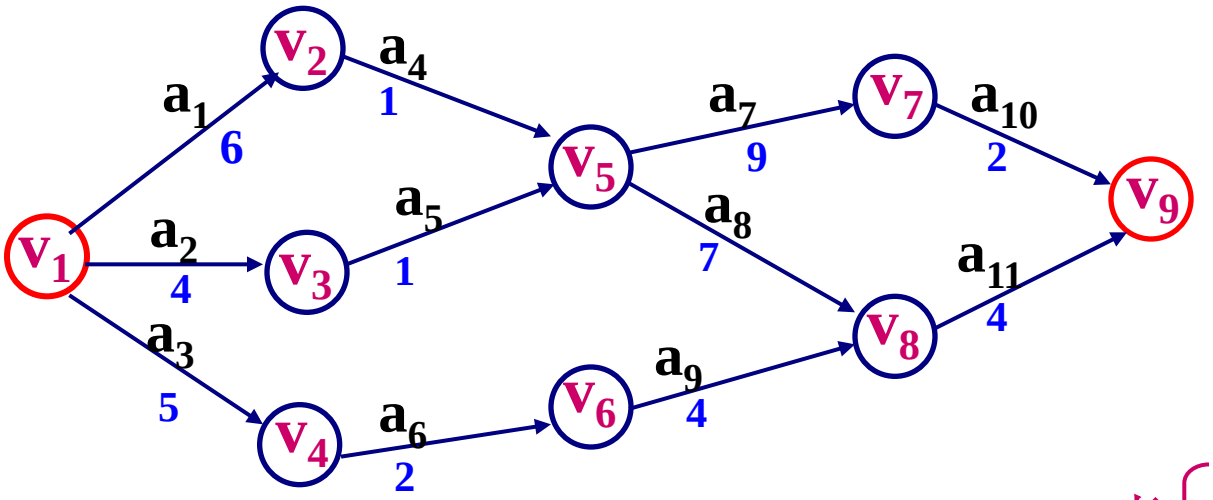
$$ee[0] = 0$$

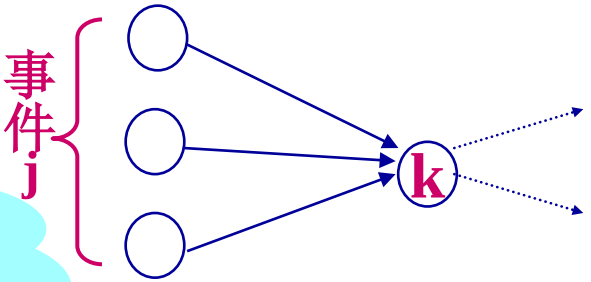
$$ee[k] = \max_{<j,k> \in P(j)} \{ ee[j] + <j,k>\text{的权} \}$$



ee[k] :

0	6	4	5	7	7	16	14	18
0	1	2	3	4	5	6	7	8





计算方法:

ee[0] = 0

ee[k] = MAX{ ee[j] + <j, k>的权}

<j,k>∈P(j)

2. 计算事件k的最晚发生时间le[k]

所谓事件k 的最晚发生时间le[k]是指不影响整个工期的前提下事件k 必须发生的最晚时间。它必须保证从事件k发出的所有活动的终点事件的最迟发生时间。

K的后继事件

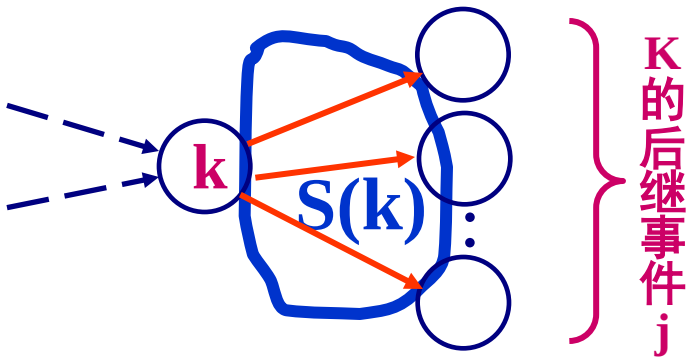
计算方法：

从后向前反推技术

$le[n-1] = ee[n-1]$

$le[k] = MIN\{ le[j] - <k,j>\text{的权} \}$

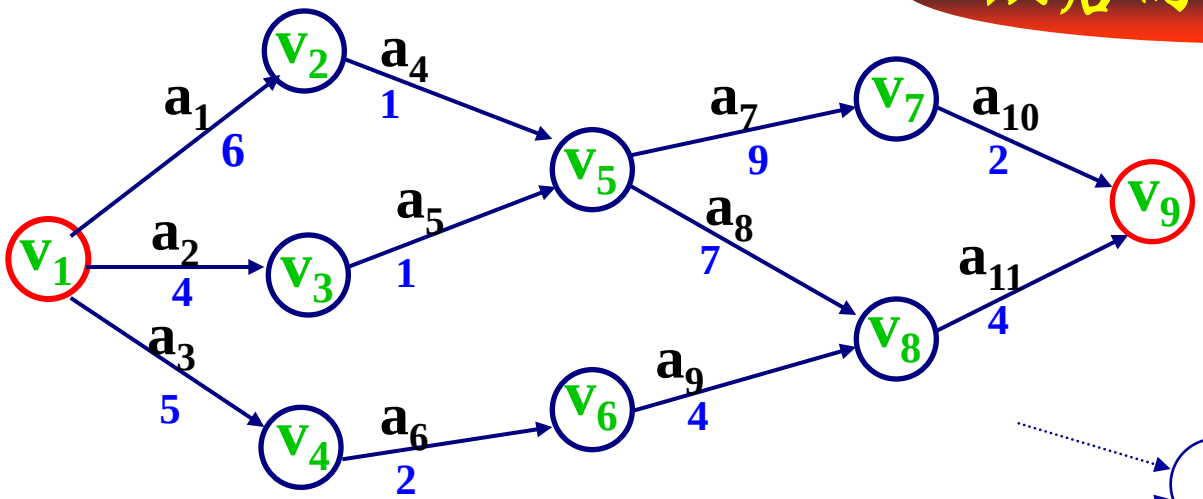
$<k,j> \in S(k)$

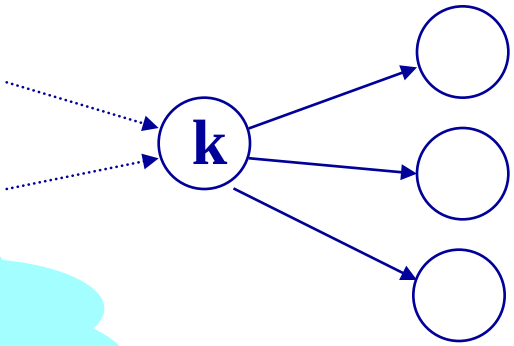


le[k] :

0	6	6	8	7	10	16	14	18
0	1	2	3	4	5	6	7	8

从后向前反推技术





计算方法：

le[n-1] = ee[n-1]

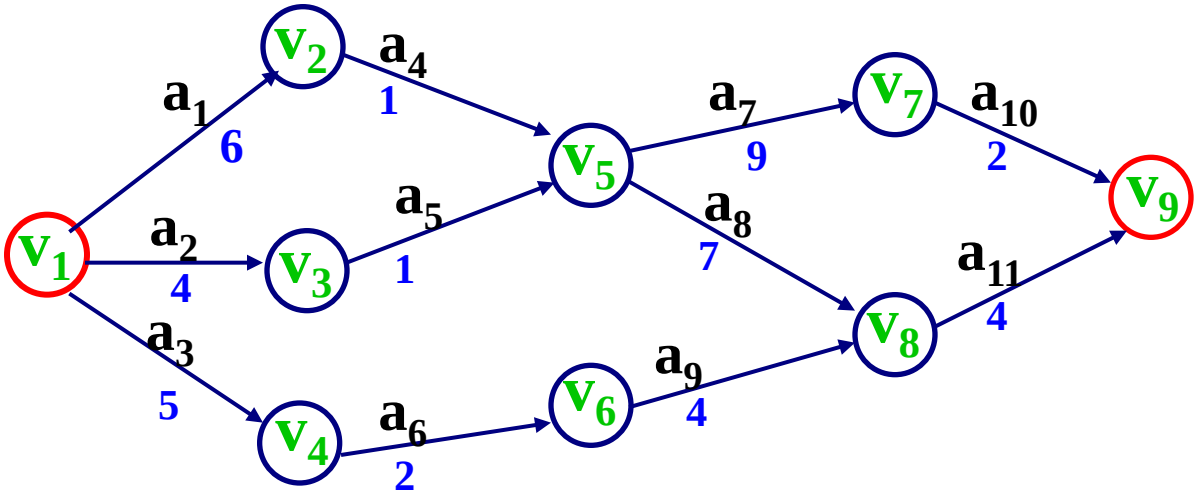
le[k] = MIN{ le[j] – <k,j>的权}

<k,j>∈S(k)

3. 计算活动 i 的最早开始时间e[i]

所谓活动i 的最早开始时间e[i] 是事件k发生的最早时间，即只有事件k 发生了，活动 i 才能开始。

计算方法：　e[i] = ee[k]



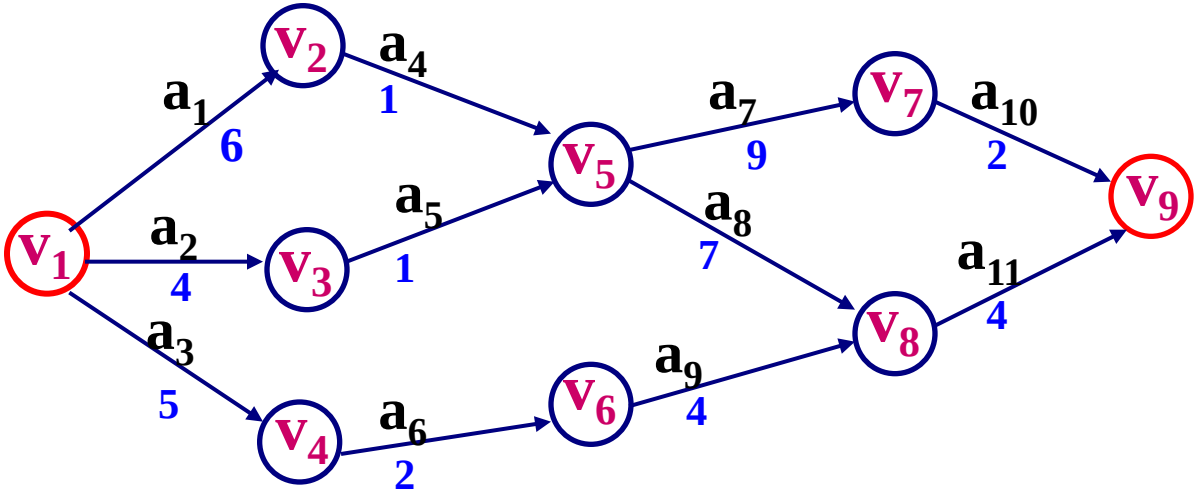
ee[k]:

0	1	2	3	4	5	6	7	8
0	6	4	5	7	7	16	14	18

(事件的最早发生时间)

e[i] :

0	1	2	3	4	5	6	7	8	9	10
0	0	0	6	4	5	7	7	7	16	14

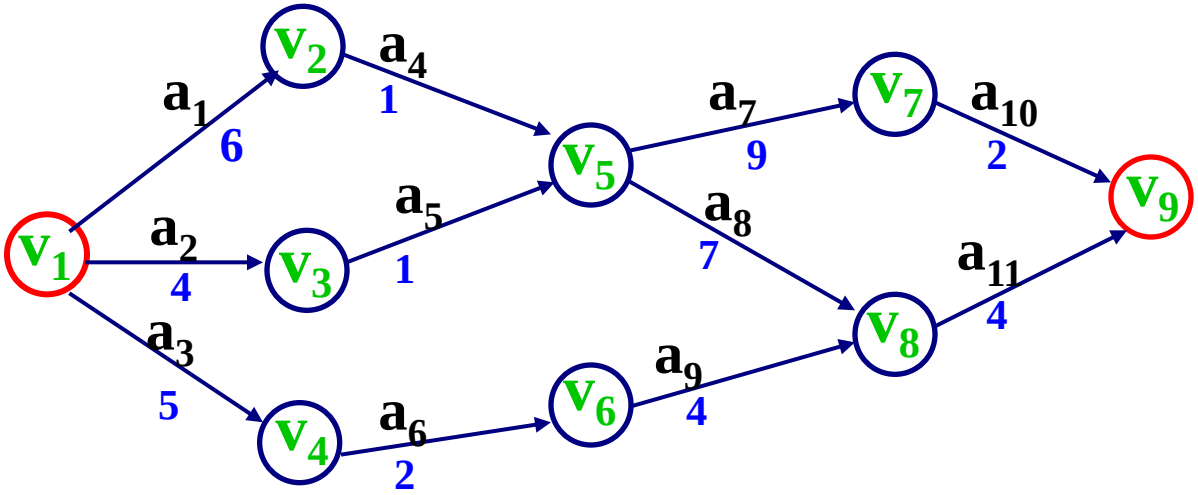


计算方法：e[i] = ee[k]

4. 计算活动 i 的最晚开始时间l[i]

所谓活动 i 的最晚开始时间l[i]是指不推迟整个工期的前提下活动 i 开始的最晚时间。

计算方法：　　l[i] = le[j] −<k,j>的权



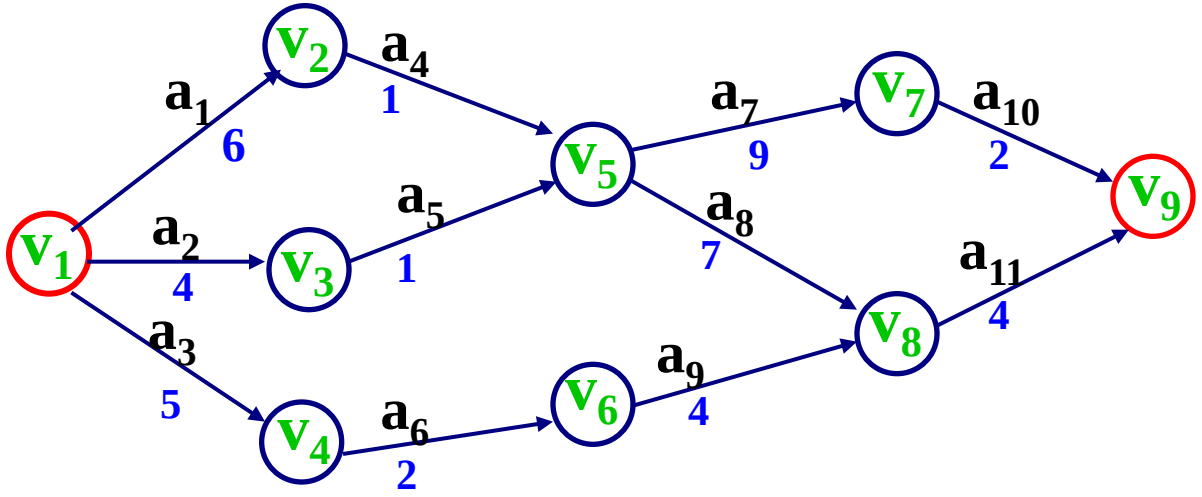
le[j] :

0	6	6	8	7	10	16	14	18
0	1	2	3	4	5	6	7	8

(事件的最晚发生时间)

l[i] :

0	2	3	6	6	8	7	7	10	16	14
0	1	2	3	4	5	6	7	8	9	10



计算方法：l[i] = le[j] −<k,j>的权

5. 求出关键活动与关键路径

计算方法：　　l[i] = e[i]

a_i是关键活动

e[i] :

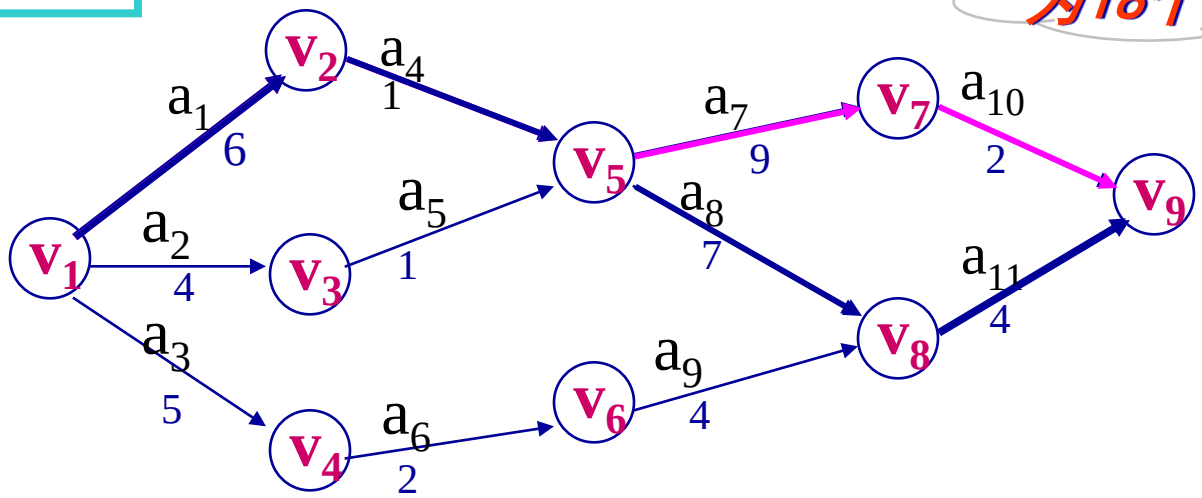
0	0	0	6	4	5	7	7	7	16	14
0	1	2	3	4	5	6	7	8	9	10

l[i] :

0	2	3	6	6	8	7	7	10	16	14
0	1	2	3	4	5	6	7	8	9	10

关键活动：a_1　a_4　a_7　a_8　a_{10}　a_{11}

关键路径的长度为18个时间单位。



思考

能否根据事件求出关键路径？

ee[k] :	0	6	4	5	7	7	16	14	18
	0	1	2	3	4	5	6	7	8
le[k] :	0	6	6	8	7	10	16	14	18
	0	1	2	3	4	5	6	7	8

否！

$v_1, v_2, v_5, v_7, v_8, v_9$

为什么？