第七章 图

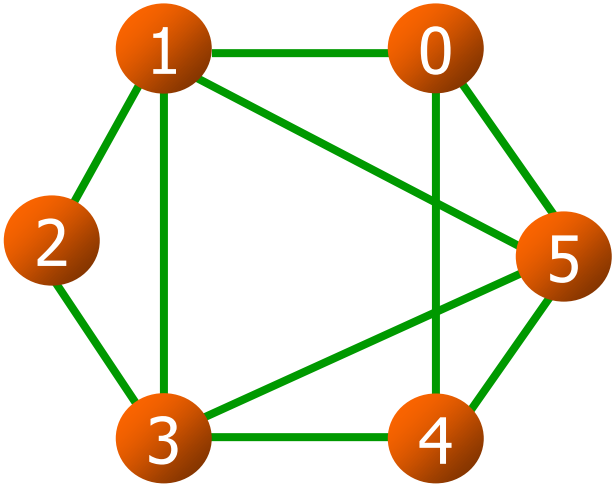
机电工程与自动化学院 L栋301

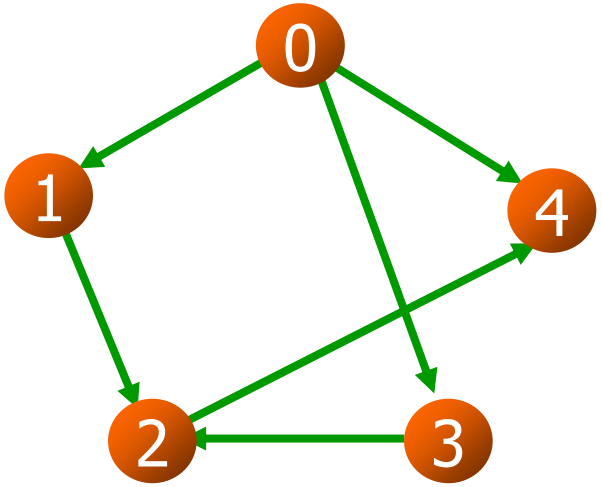
任卫红 助理教授

renweihong@hit.edu.cn

http://faculty.hitsz.edu.cn/renweihong

第7章　图





7.1图的定义与术语

第一节　图的定义与术语

一、图的定义(Graph)

- 图是由顶点集合(vertex)及顶点间的关系集合组成的一种数据结构：

Graph＝(V, E)

其中V = {x | x∈数据对象}是顶点的有穷非空集合

E是顶点之间关系的有穷集合，包括

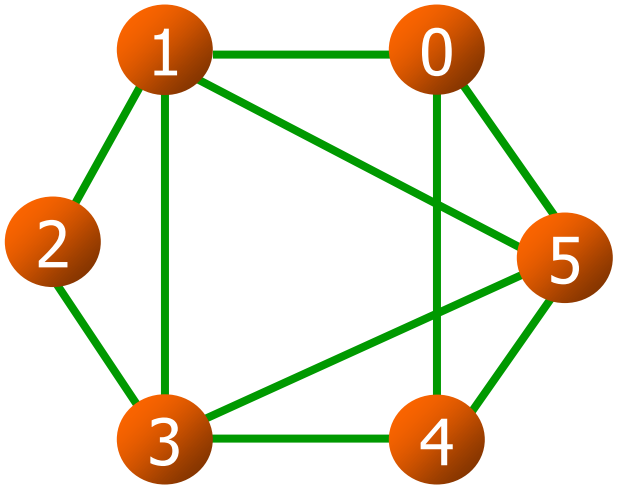
E1 = {(x, y) | x, y ∈ V } 边的集合

或　E2 = {<x, y> | x, y ∈ V } 弧的集合

第一节　图的定义与术语

二、无向图(Undigraph)

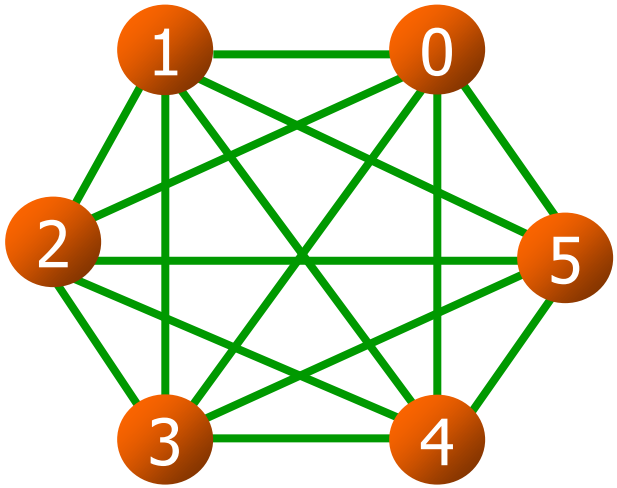
- 用(x, y)表示两个顶点x, y之间的一条边(edge)
- N={V, E}，V={0, 1, 2, 3, 4, 5}，E={(0, 1)，(0, 4)，(0, 5)，(1, 2)，(1, 3)，(1, 5)，(2, 3)，(3, 4)，(3, 5)，(4, 5)}



第一节　图的定义与术语

二、无向图(完全图)

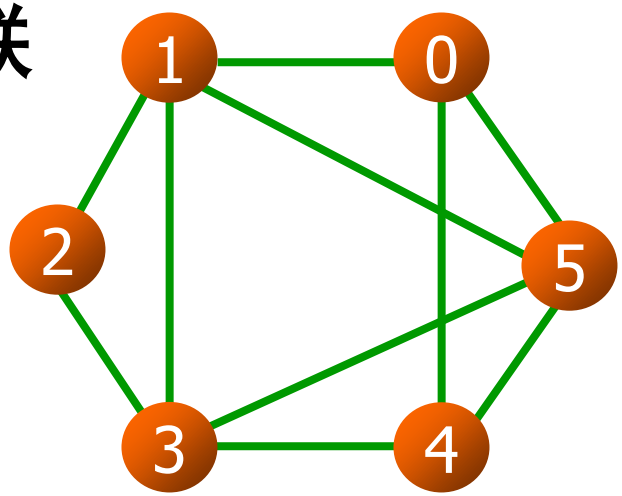
- 如果无向图有n(n-1)/2条边，则称为无向完全图



第一节　图的定义与术语

二、无向图

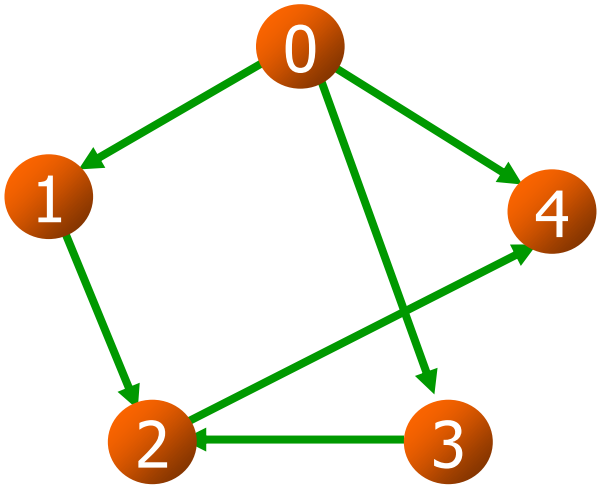
- **邻接点**：如果(x, y)∈E, 称x, y互为邻接点，即x, y相邻接
- **依附**：边(x, y)依附于顶点x, y
- **相关联**：边(x, y)与x, y相关联
- **顶点的度**：和顶点相关联的边的数目，记为TD(x)



第一节　图的定义与术语

三、有向图(Digraph)

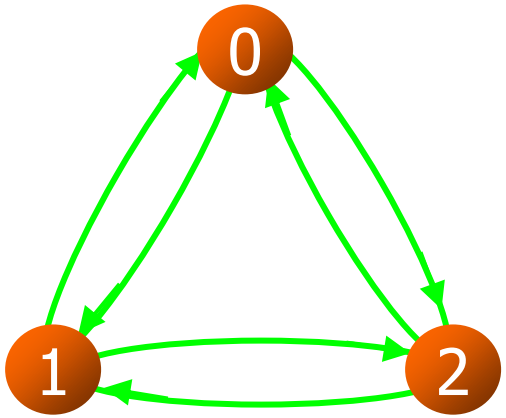
- 用<x,y>表示从x到y的一条弧(Arc)，且称x为弧尾，y为弧头，
- N={V,E}，V={0,1,2,3,4}，E={<0,1>，<0,3>，<0,4>，<1,2>，<2,4>，<3,2> }



第一节　图的定义与术语

三、有向图(完全图)

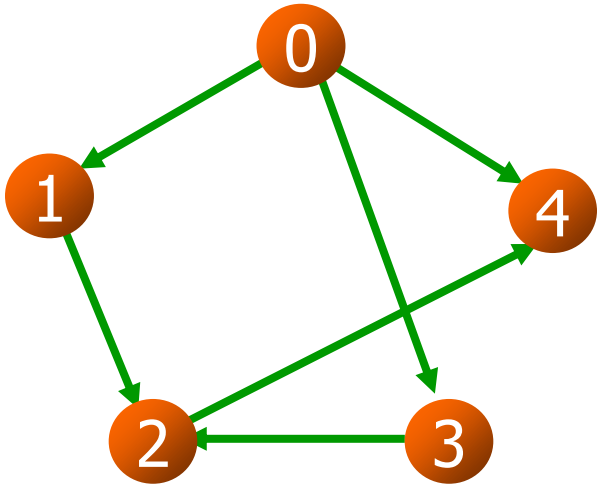
- 如果有向图有n(n-1)条边，则称为有向完全图



第一节　图的定义与术语

三、有向图

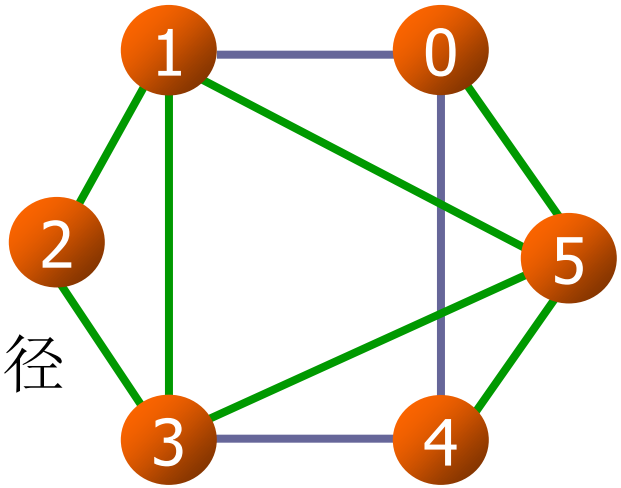
- **邻接**：如果<x, y>∈E, 称x邻接到y, 或y邻接自x
- **相关联**：弧<x, y>与x, y相关联
- **入度**：以顶点为头的弧的数目，记为ID(x)
- **出度**：以顶点为尾的弧的数目，记为OD(x)
- **度**：TD(x)=ID(x)+OD(x)



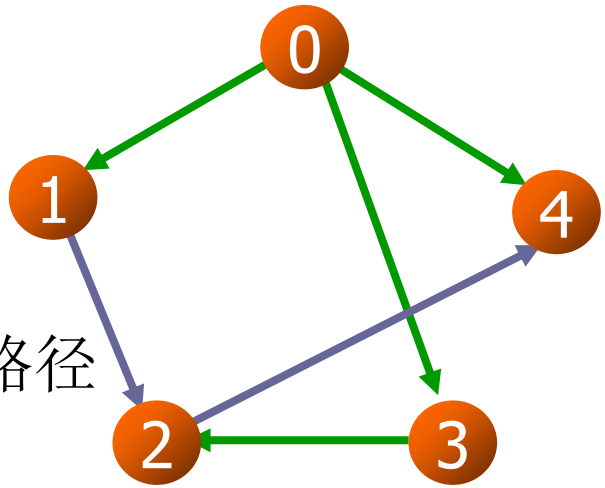
第一节　图的定义与术语

四、路径(Path)

- **路径**：是一个从顶点x到y的顶点序列(x，v_{i1}，v_{i2}，…，v_{in}，y)
- 其中，(x, v_{i1})，(v_{ij-1}, v_{ij})，(v_{in}, y)皆属于E



1到3有路径
(1,0,4,3)

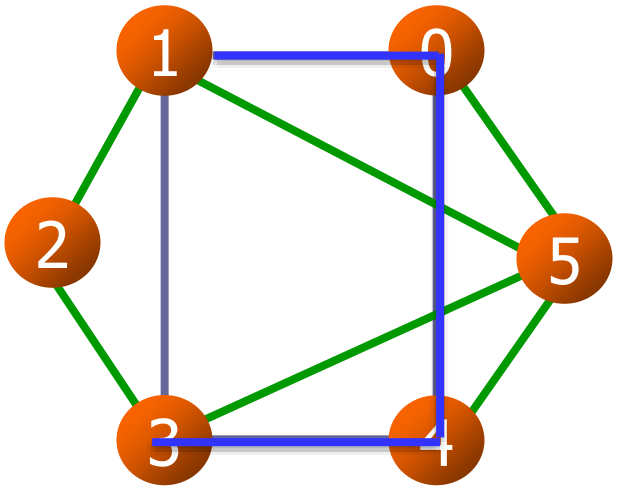


1到4有路径
(1,2,4)

第一节　图的定义与术语

五、回路

- **回路或环**：路径的开始顶点与最后一个顶点相同，即路径中（x，v_{i1}，v_{i2}，…，v_{in}，y），x=y
- **简单路径**：路径的顶点序列中，顶点不重复出现



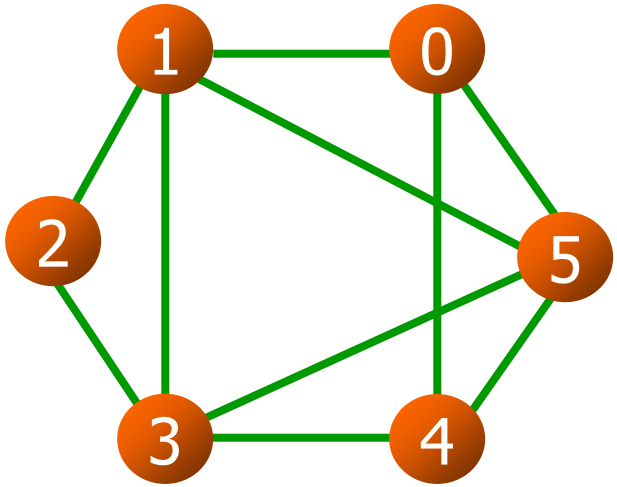
1到1构成环
(1,0,4,3,1)

1到3是简单路径
(1,0,4,3)

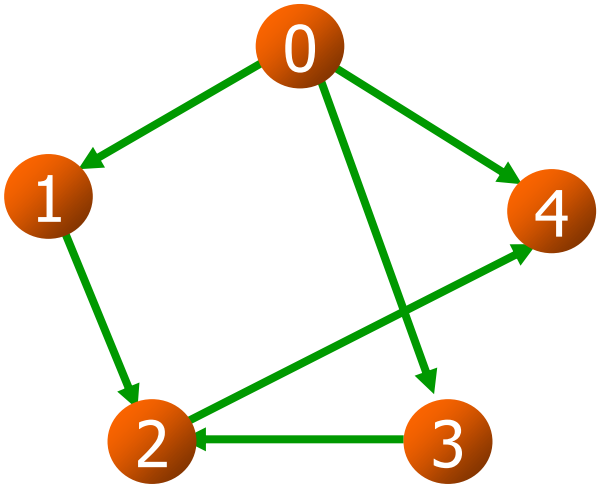
第一节　图的定义与术语

六、连通

- **连通**：如果顶点x到y有路径，称x和y是连通的
- **连通图**：图中所有顶点都连通



连通图

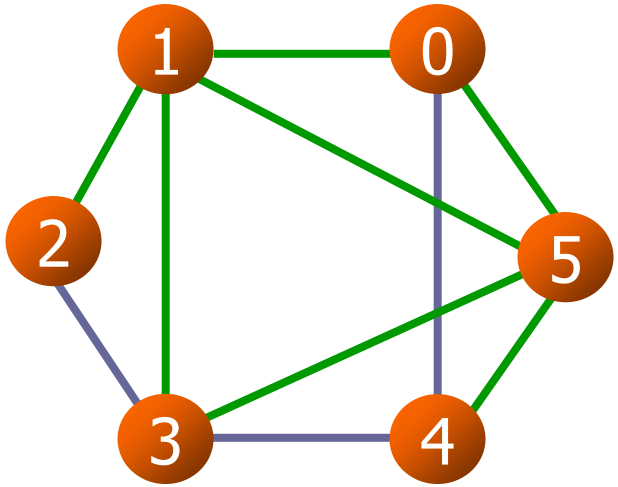


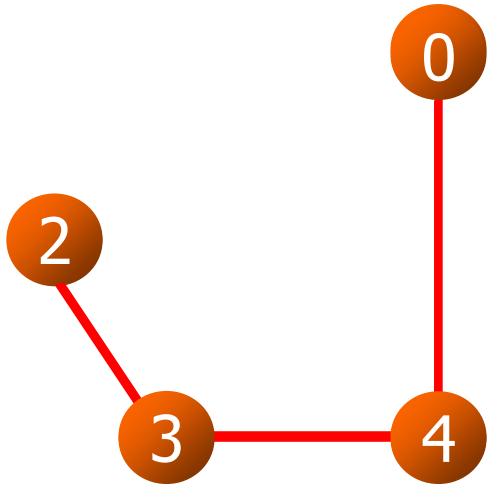
非连通图

第一节　图的定义与术语

七、子图

- 设有两个图 G＝(V, E) 和 G'＝(V', E')。若 V'⊆ V 且 E'⊆E, 称图G'是图G的子图

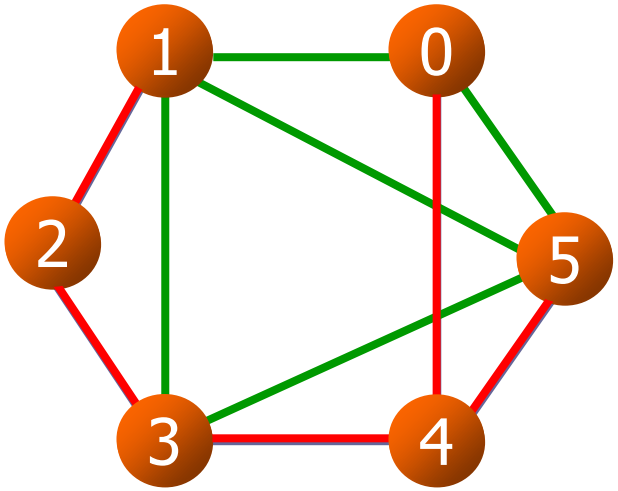




第一节　图的定义与术语

八、生成树

- 一个连通图的生成树是一个极小连通子图，它含有图中全部n个顶点，但只有足以构成一棵树的n-1条边



7.2 图的存储结构

第二节　图的存储结构

一、邻接矩阵(Adjacency Matrix)

- 邻接矩阵：记录图中各顶点之间关系的二维数组。
- 对于不带权的图，以1表示两顶点存在边(或弧)(相邻接)，以0表示两顶点不邻接，即

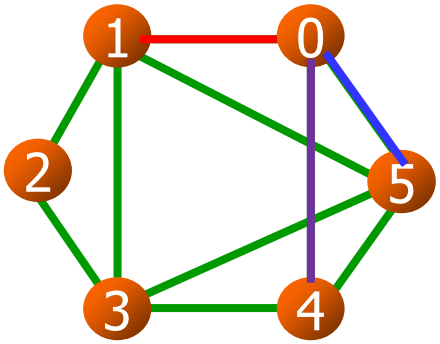
$$A[i][j] = \begin{cases} 1 & \text{如果}(i,j)\in E \text{ 或 } \langle i,j\rangle\in E \\ 0 & \text{其它} \end{cases}$$

第二节　图的存储结构

一、邻接矩阵

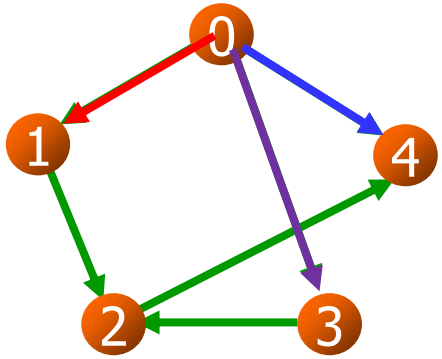
- 无向图的邻接矩阵为：

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



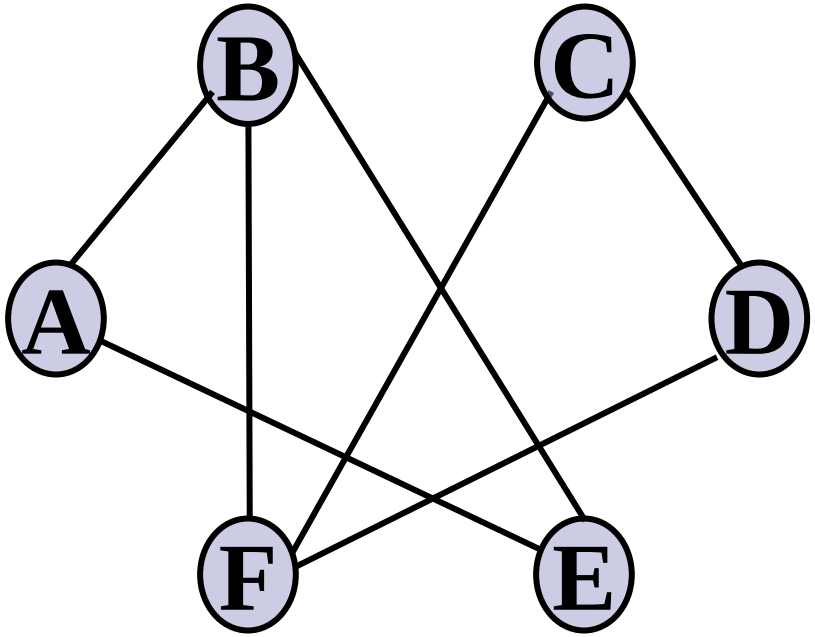
- 有向图的邻接矩阵为：

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



第二节　图的存储结构

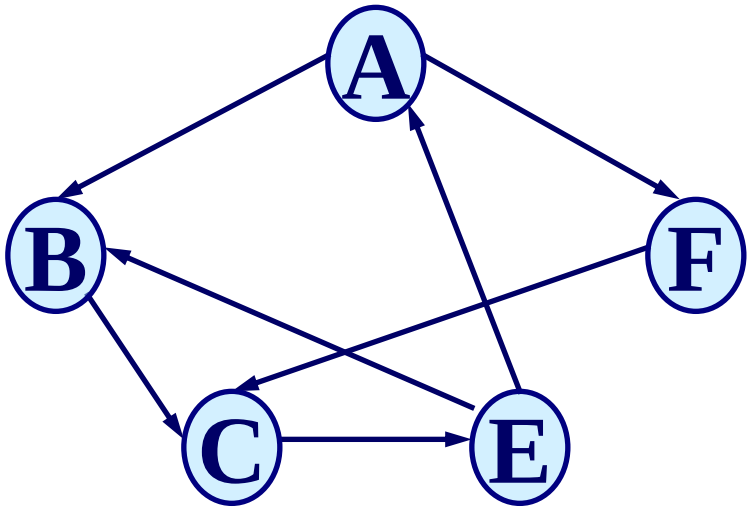
练习：写出下列图的邻接矩阵表示。



	A	B	C	D	E	F
A	0	1	0	0	1	0
B	1	0	0	0	1	1
C	0	0	0	1	0	1
D	0	0	1	0	0	1
E	1	1	0	0	0	0
F	0	1	1	1	0	0

无向图邻接矩阵为对称阵。

第二节　图的存储结构



	A	B	C	E	F
A	0	1	0	0	1
B	0	0	1	0	0
C	0	0	0	1	0
E	1	1	0	0	0
F	0	0	1	0	0

有向图的邻接矩阵为非对称矩阵。

第二节　图的存储结构

一、邻接矩阵(性质)

- 无向图的邻接矩阵是对称的
  - 其第i行1的个数或第i列1的个数，等于顶点i的度TD(i)

- 有向图的邻接矩阵可能是不对称的
  - 第i行1的个数等于顶点i的出度OD(i)
  - 第j列1的个数等于顶点j的入度ID(j)

第二节　图的存储结构

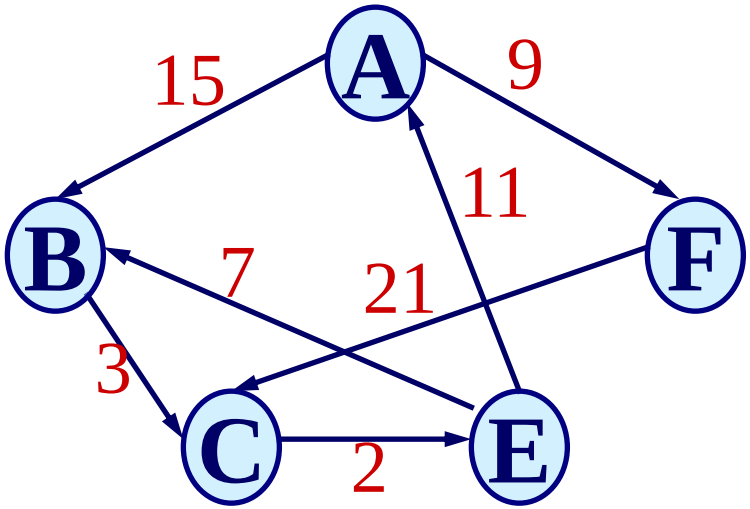
一、邻接矩阵(网络)

- 在网络中，两个顶点如果不邻接，则被视为距离为无穷大；如果邻接，则两个顶点之间存在一个距离值(即权值)

$$A[i][j] = \begin{cases} w_{i,j} & \text{如果}(i,j)\in E \text{ 或 } \langle i,j\rangle \in E \\ \infty & \text{其它} \end{cases}$$

第二节　图的存储结构

练习：若弧上加权，原邻接矩阵如何修改？



∞	15	∞	∞	9
∞	∞	3	∞	∞
∞	∞	∞	2	∞
11	7	∞	∞	∞
∞	∞	21	∞	∞

第二节　图的存储结构

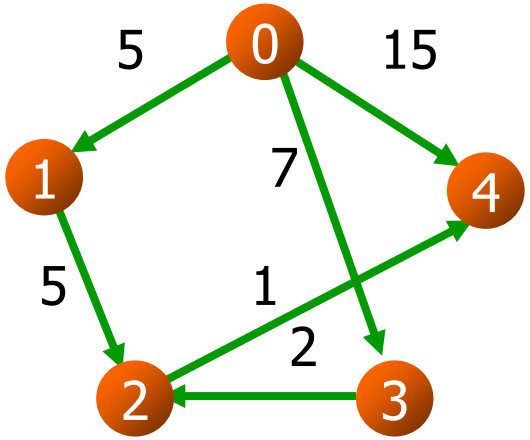
一、邻接矩阵(网络)

作业

- 有向网N={V, E}，V={0, 1, 2, 3, 4}，E={<0, 1, 5>，<0, 3, 7>，<0, 4, 15>，<1, 2, 5>，<2, 4, 1>，<3, 2, 2> }，E中每个元组的第三个元素表示权。

1、画出该网，2、写出该网的邻接矩阵。

$$\begin{pmatrix} \infty & 5 & \infty & 7 & 15 \\ \infty & \infty & 5 & \infty & \infty \\ \infty & \infty & \infty & \infty & 1 \\ \infty & \infty & 2 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{pmatrix}$$



第二节　图的存储结构

```
class MGraph{
    int     Vexnum;// 顶点数
    int     Arcnum; // 边数
    char   *Vexs;     //顶点信息集，string  *Vexs;
    int    **Edges;    //边信息集
public:
    MGraph(int n);  //构造函数，赋值并分配空间
    ~MGraph();
}
```

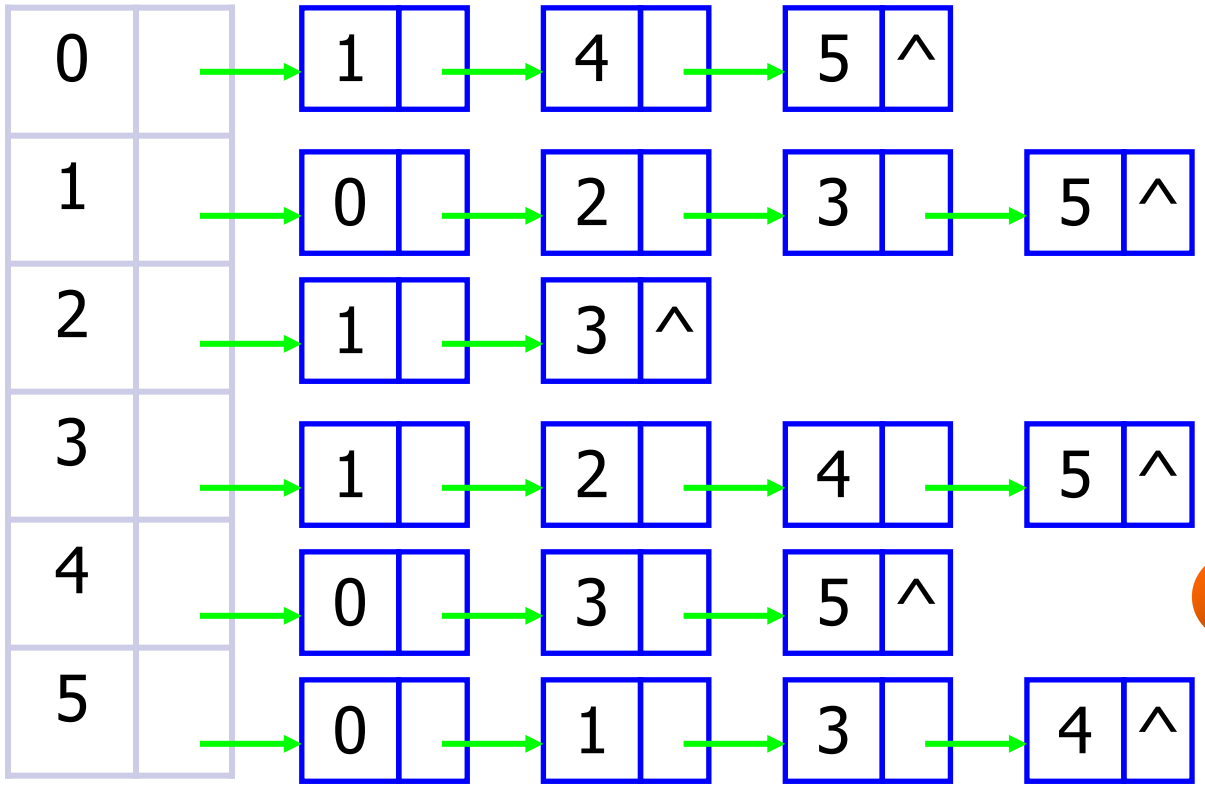
第二节　图的存储结构

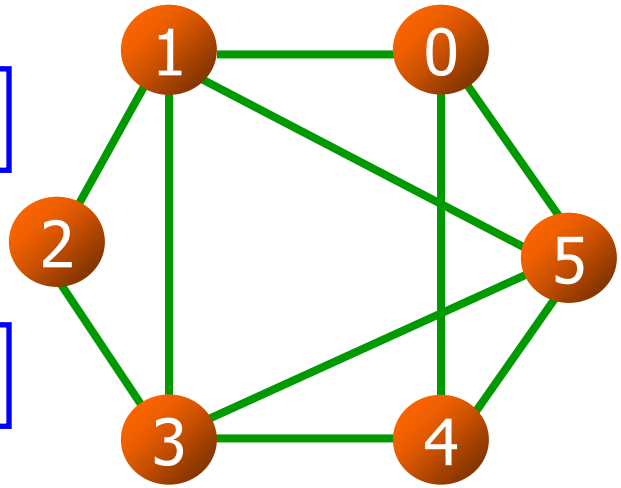
二、邻接表(Adjacency List)

- 邻接表是图的一种链式存储结构
- 在邻接表中，每个顶点设置一个**单链表**，其每个结点都是**依附于该顶点的边**（或以该顶点为尾的弧）

第二节　图的存储结构

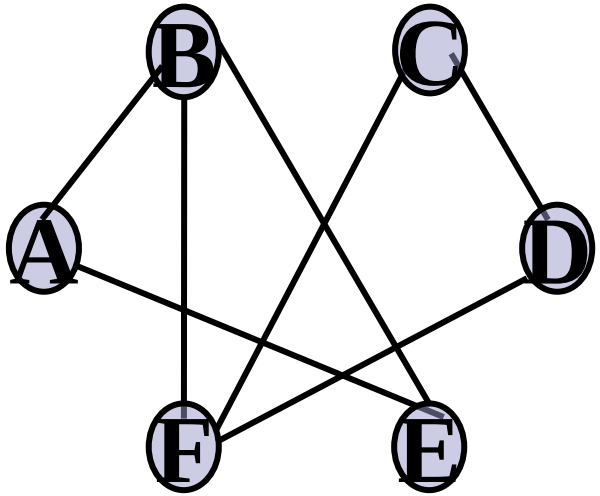
二、邻接表(无向图)

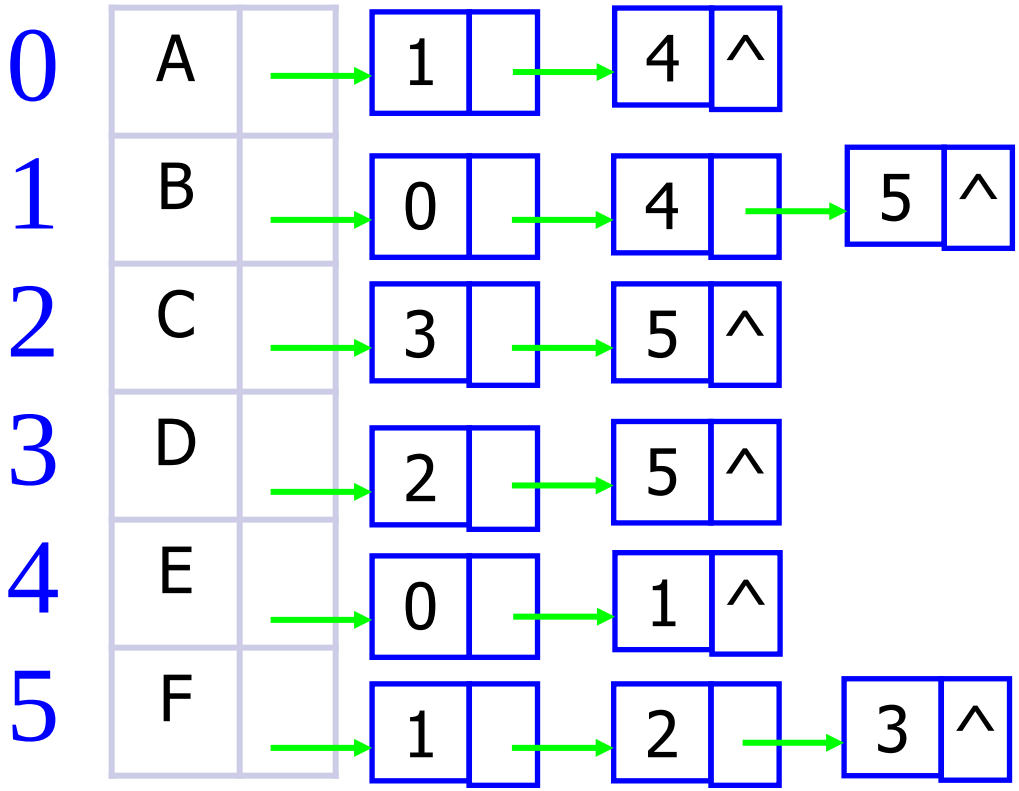




第二节　图的存储结构

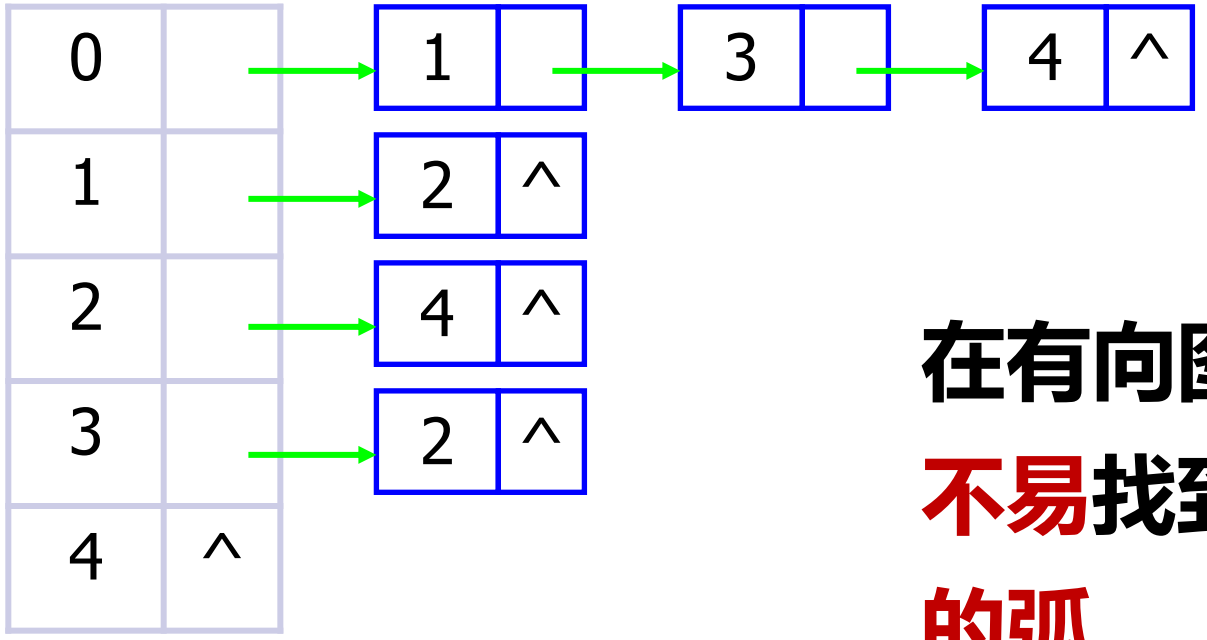
练习：画出右图的邻接表表示。

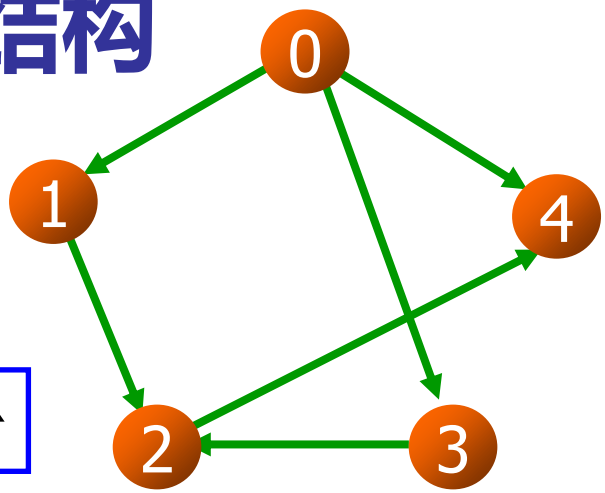




第二节　图的存储结构

二、邻接表(有向图)

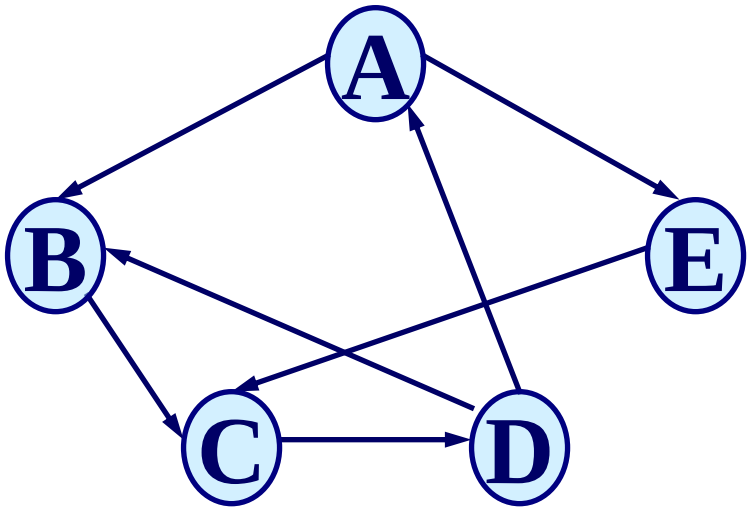


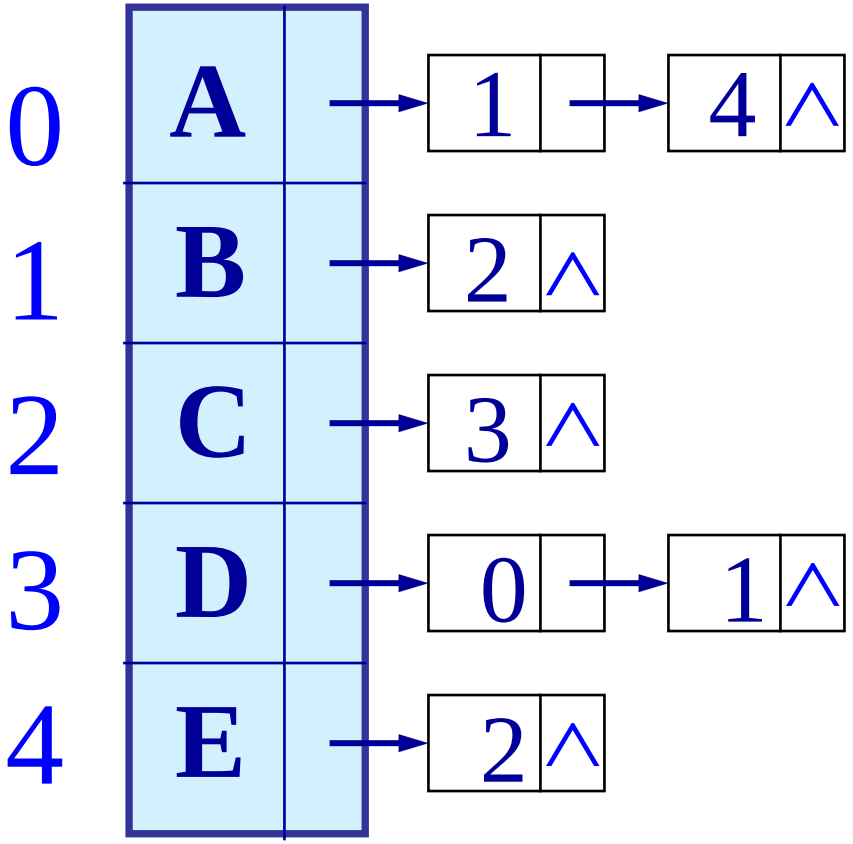


在有向图的邻接表中不易找到指向该顶点的弧。

第二节　图的存储结构

练习：画出左图的邻接表表示。

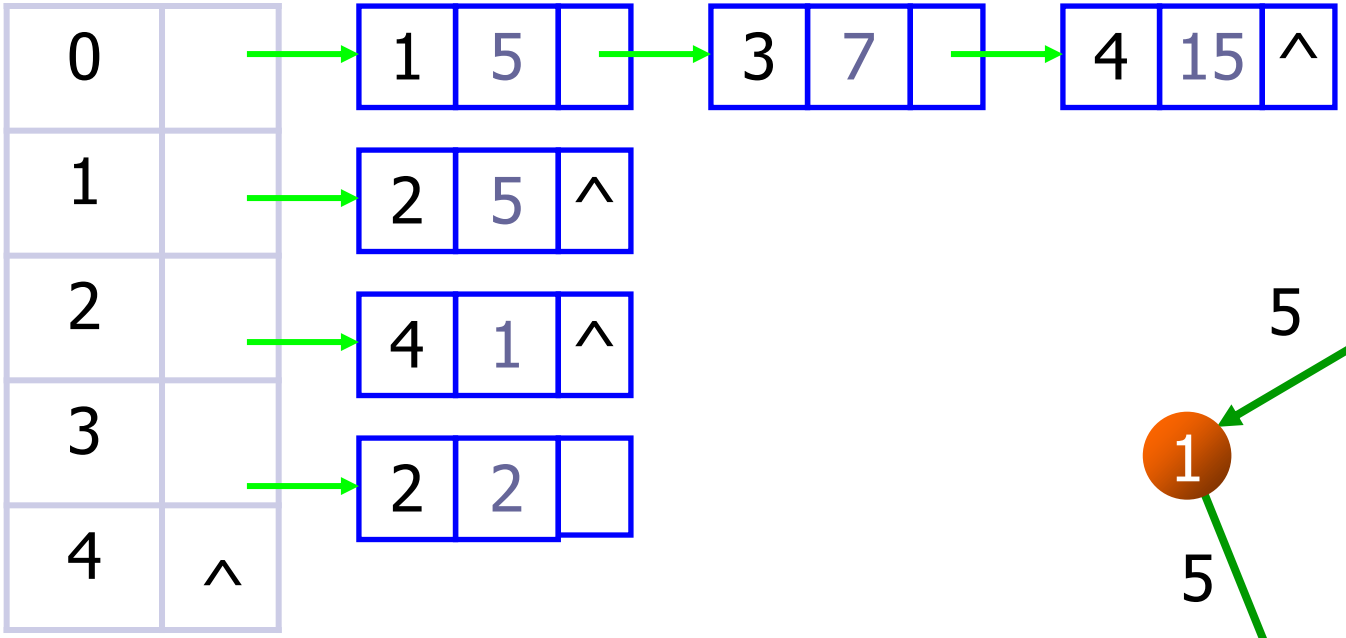


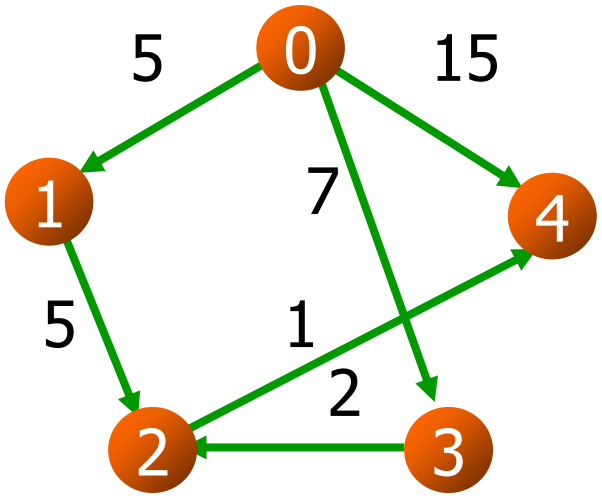


第二节　图的存储结构

二、邻接表(网络)

作业

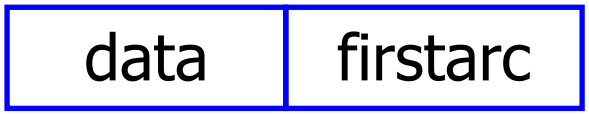




第二节　图的存储结构

二、邻接表(结点结构)

- 顶点的结点结构



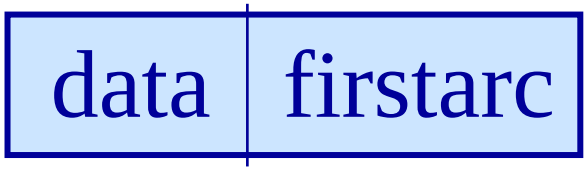

```
data;      // 顶点信息
firstarc;//指向第一条依附该顶点的边(弧)
```

- 边(弧)的结点结构


```
adjvex; // 该边(弧)所指向的顶点的位置
nextarc;// 指向下一条边(弧)指针
info;   // 该边(弧)相关信息的指针或权值
```

第二节　图的存储结构

顶点的结点结构




```
class  VNode {
    char  data;   // 顶点信息
    ArcNode  *firstarc;
                  // 指向第一条依附该顶点的弧
  };
```

第二节　图的存储结构

弧的结点结构


```
class  ArcNode {
   int             adjvex;   // 该弧所指向的顶点的位置
   ArcNode    *nextarc;
                        // 指向下一条弧的指针
   InfoType   *info;   // 该弧相关信息的指针,或权值
};
```

第二节　图的存储结构

图的结构定义

```
class  ALGraph{
    VNode  *vertices;
    int      vexnum, arcnum;
    //int      kind;          // 图的种类标志
 };
```

第二节　图的存储结构

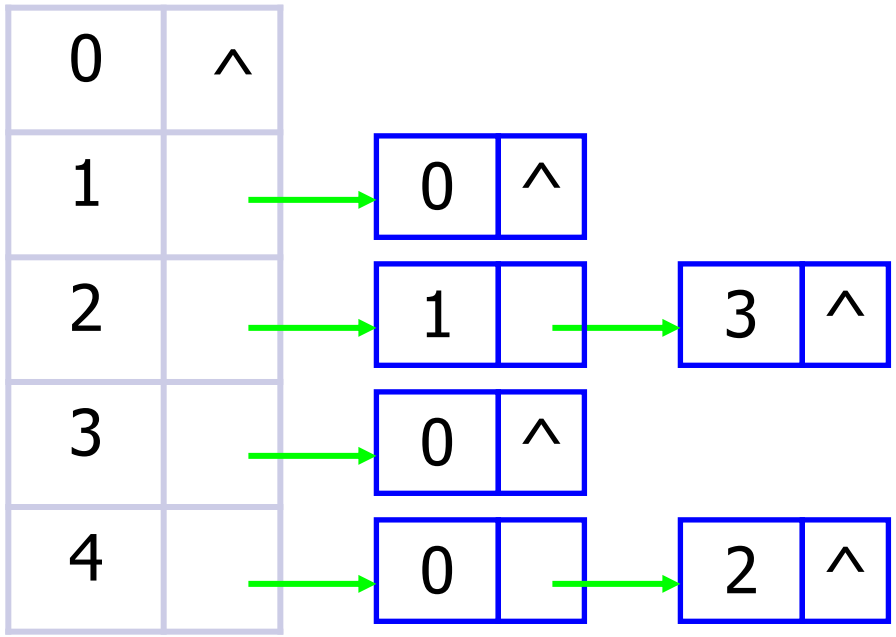
二、邻接表(性质)

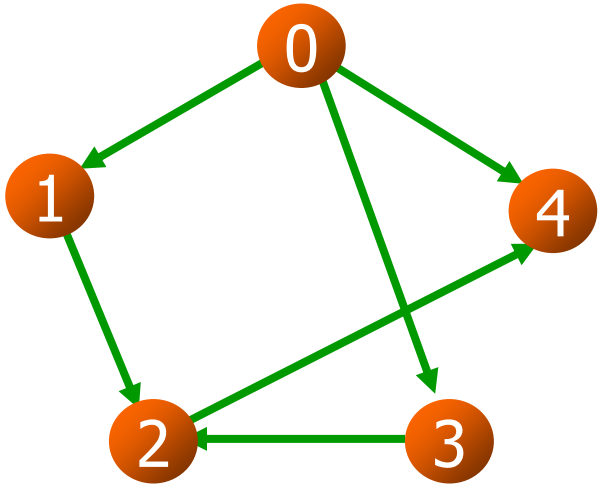
- 对于有向图的邻接表，其第i个链表中结点的个数只是该顶点的出度；如果要计算入度，必须遍历整个邻接表[也可以建立一个逆邻接表]
- 要判定两个顶点i和j是否有边（或弧），必须搜索整个第i个和第j个链表，不及邻接矩阵方便

第二节　图的存储结构

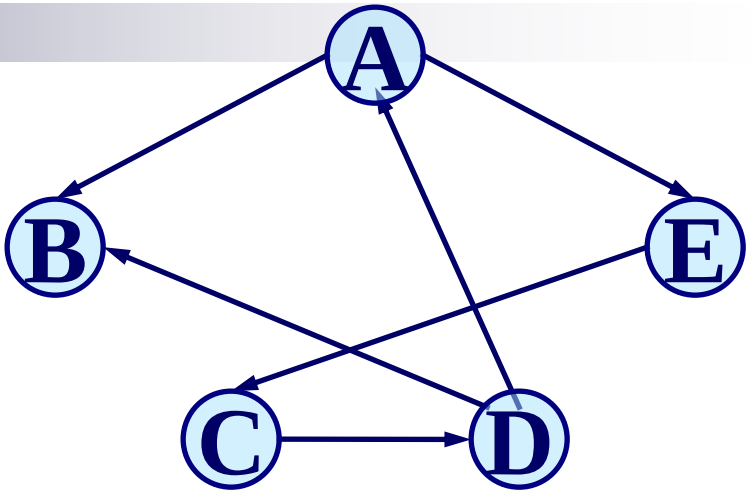
二、邻接表(有向图的逆邻接表)

- 逆邻接表中，弧的箭头向内(入弧)

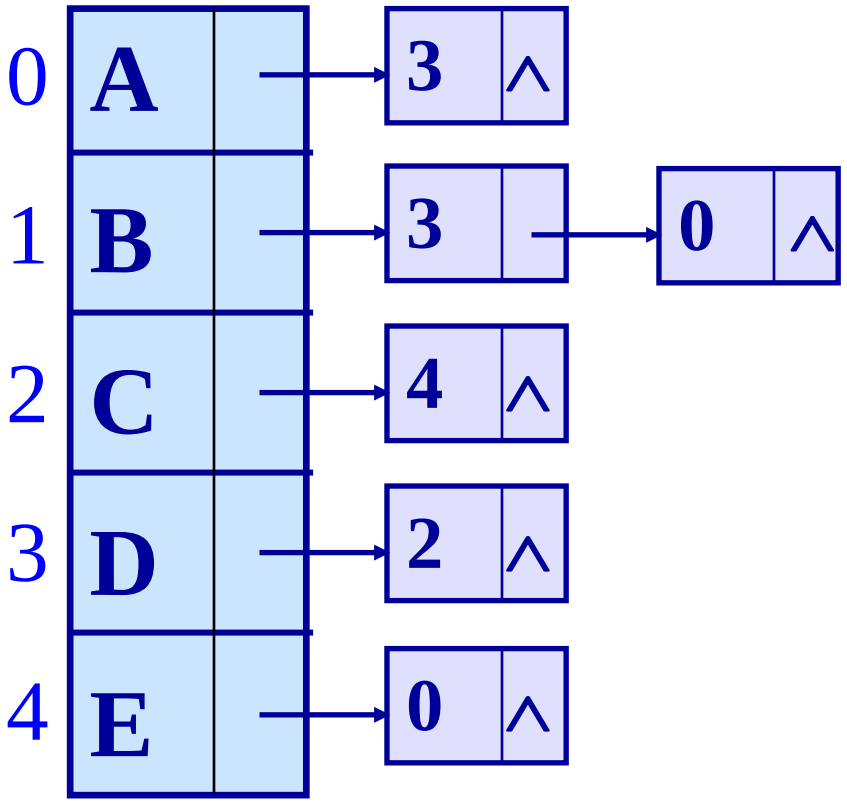




有向图的逆邻接表



有向图的逆邻接表中，每个顶点，链接的是指向该顶点的弧。（以该顶点为头的弧）



第二节　图的存储结构

三、十字链表(Orthogonal List)

- 十字链表是有向图的另一种存储结构
- 十字链表是将有向图的邻接表和逆邻接表结合起来的一种存储结构

第二节　图的存储结构

三、十字链表(结点结构)

- 弧的结点结构

```
tailvex;// 弧尾顶点的位置
headvex;// 弧头顶点的位置
hlink;  // 指向弧头相同的下一条弧
tlink;  // 指向弧尾相同的下一条弧
info;   // 该弧相关信息的指针或权值
```

tailvex	headvex	hlink	tlink	info

第二节　图的存储结构

三、十字链表(结点结构)

- 顶点的结点结构

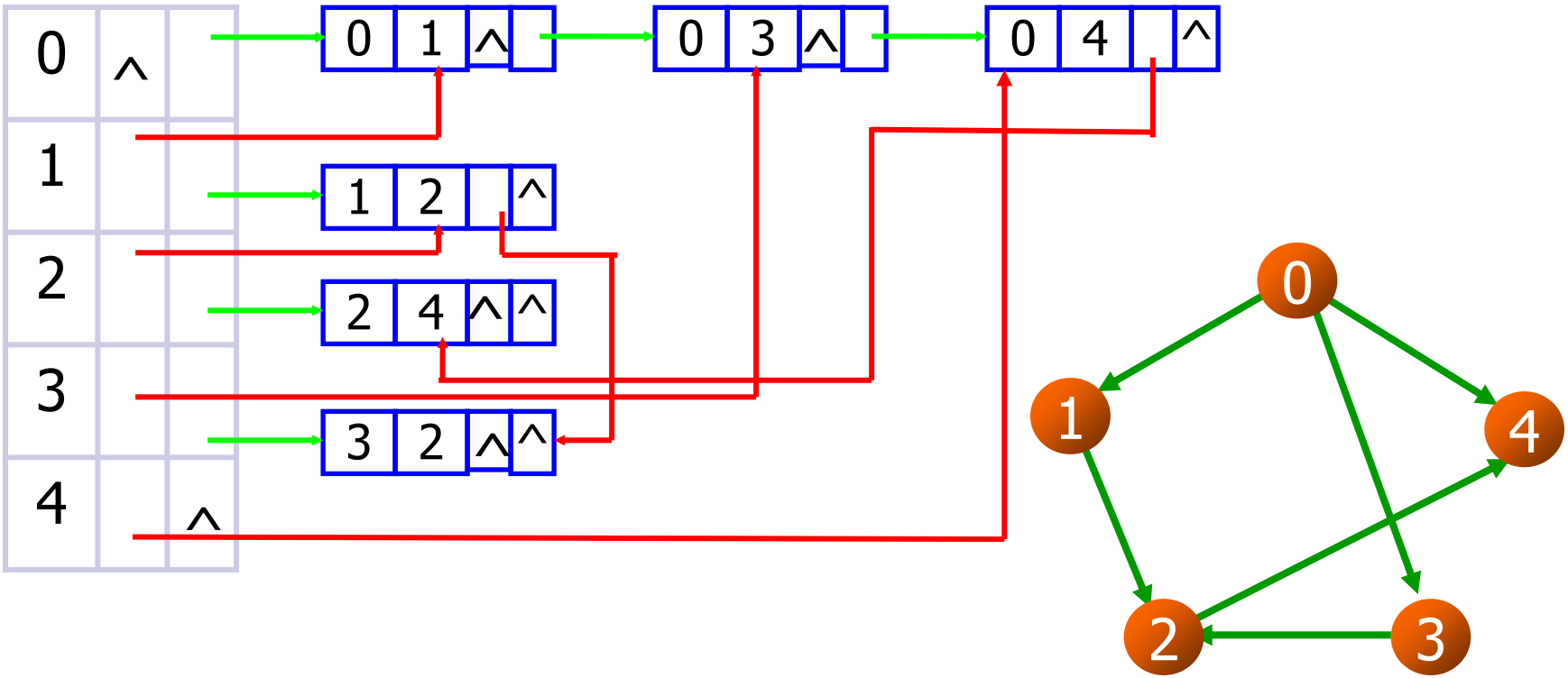
```
data;     // 与顶点相关的信息
firstin; // 指向以顶点为弧头的第一个弧结点
firstout;// 指向以顶点为弧尾的第一个弧结点
```

第二节　图的存储结构

三、十字链表(举例)

相同弧尾（邻接表）

相同弧头（逆邻接表）



第二节　图的存储结构

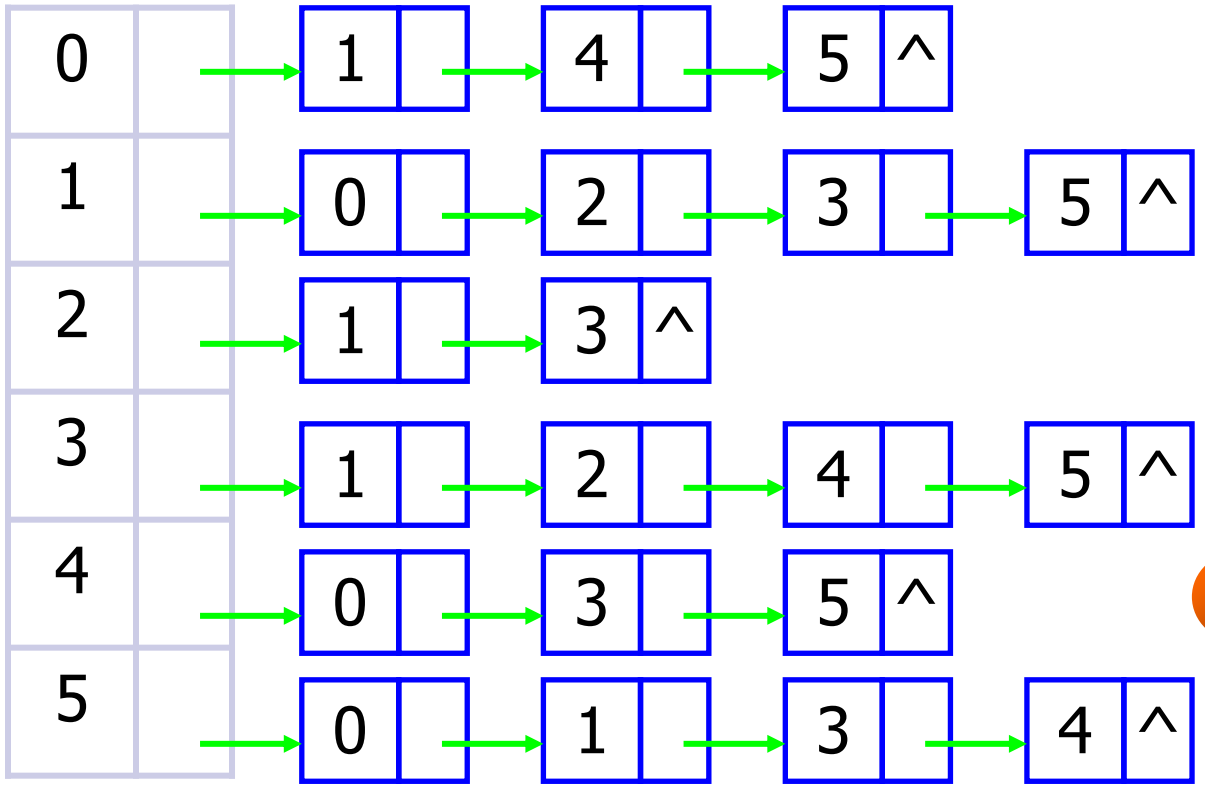
四、邻接多重表(Adjacency Multilist)

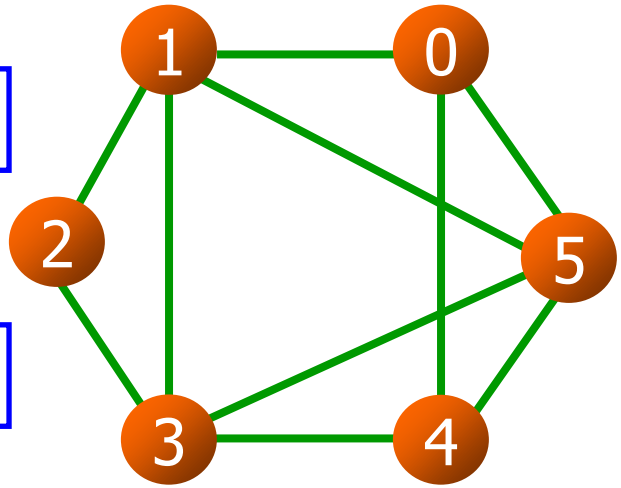
- 邻接多重表是无向图的另一种存储结构
- 在无向图中，一条边要用2个结点表示(分别从2个顶点的角度看)

- 在邻接多重表中，一条边只用一个结点表示
- 将所有具有某顶点的结点，全部用链连结起来，链所在的域为该顶点对应的指针域

第二节　图的存储结构

四、邻接多重表(Adjacency Multilist)





总共10条变，但是实际使用20个边节点表示。

第二节　图的存储结构

四、邻接多重表(结点结构)

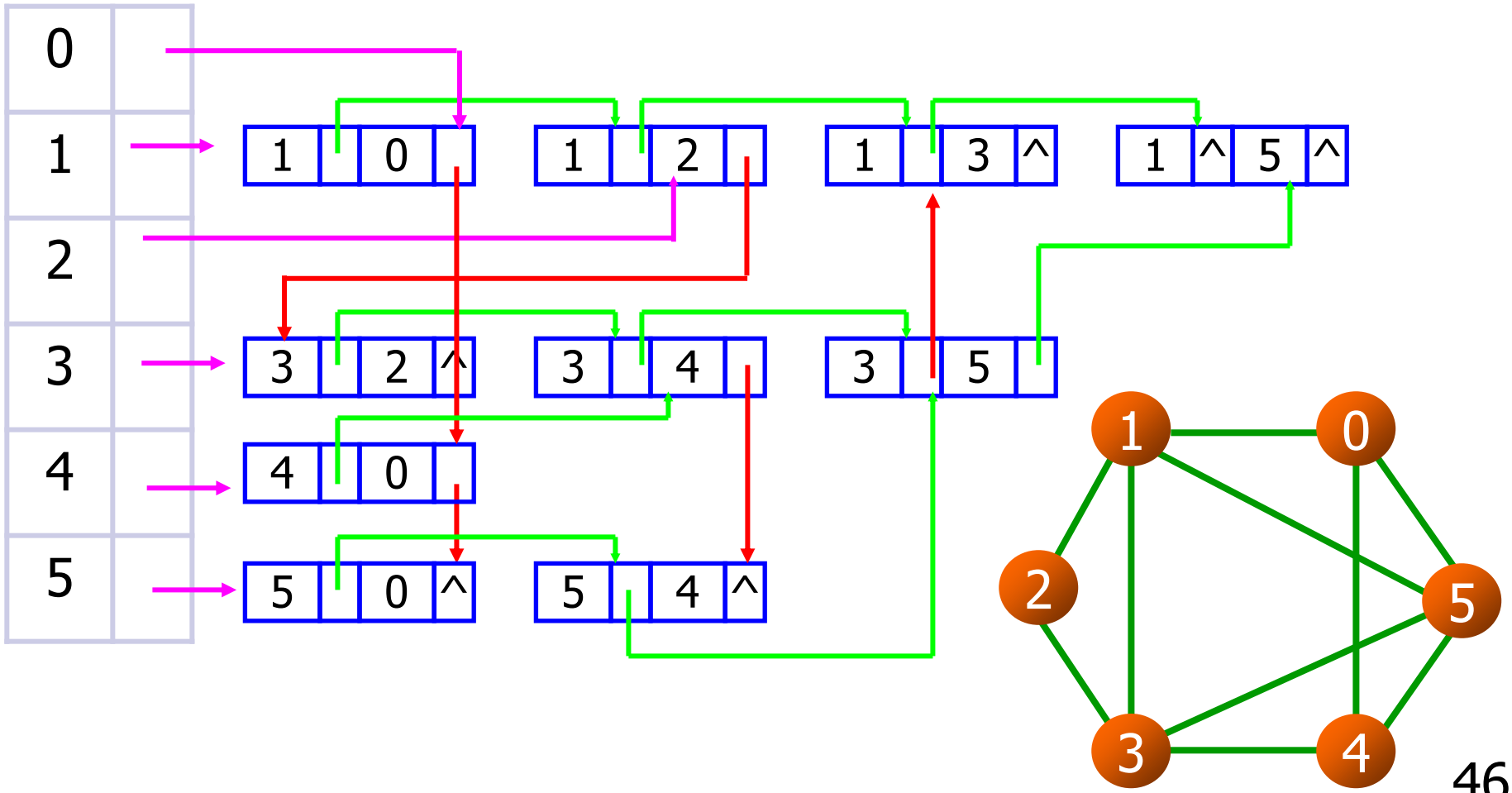
- 边的结点结构

  mark; // 标记域，如指示该边是否被搜索过
  ivex, jvex;// 该边所依附的两个顶点的位置
  ilink;// 指向下一条依附于ivex的边
  jlink;// 指向下一条依附于jvex的边
  info; // 该边相关信息的指针或权值

mark	ivex	ilink	jvex	jlink	info

第二节　图的存储结构

四、邻接多重表(举例)



7.3 图的遍历

第三节　图的遍历

一、图的遍历

- 从图的某一顶点开始，访遍图中其余顶点，且使每一个顶点仅被访问一次
- 图的遍历主要应用于无向图

第三节　图的遍历

二、深度优先搜索(DFS)

- 图的深度优先搜索是**树的先根遍历**的推广
- 图中可能存在回路，且图的任一顶点都可能与其它顶点相通，在访问完某个顶点之后可能会沿着某些边又回到了曾经访问过的顶点。
- 为了**避免重复访问**，可设置一个标志顶点是否被访问过的辅助数组 visited[ ]

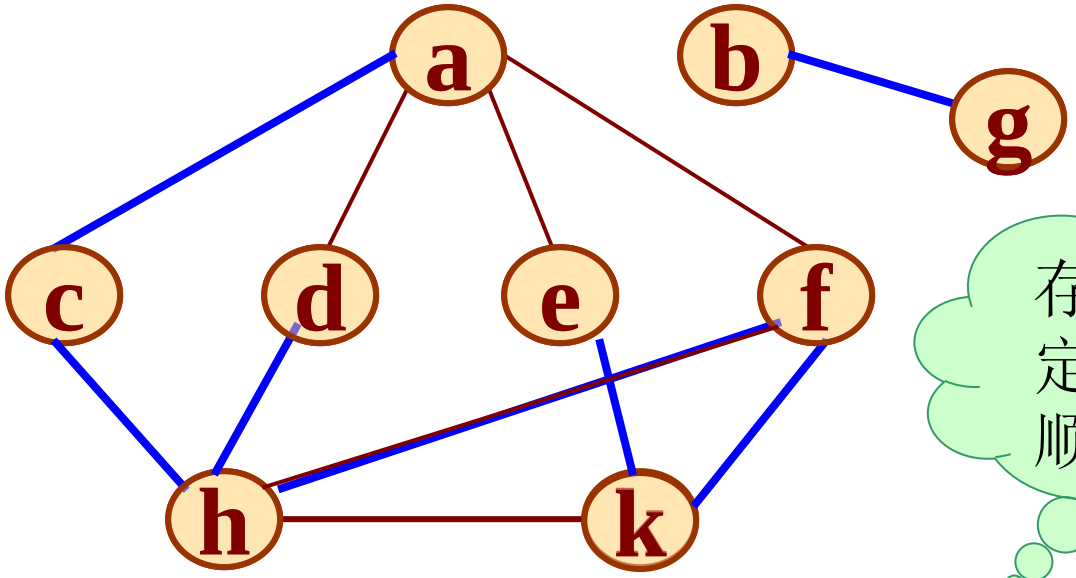
第三节　图的遍历

二、深度优先搜索(DFS算法)

- 所有顶点访问标志visited[]设置为FALSE
- 从某顶点v_0开始，设$v=v_0$

1. 如果visited[v]=FALSE，则访问该顶点，且设visited[v]=TRUE
2. 如果找到当前顶点的一个新的相邻顶点w，设v=w，重复1
3. 否则(说明当前顶点的所有相邻顶点都已被访问过，或者当前顶点没有相邻顶点)，如果当前顶点是v_0，退出；否则返回上一级顶点，重复2

例如:



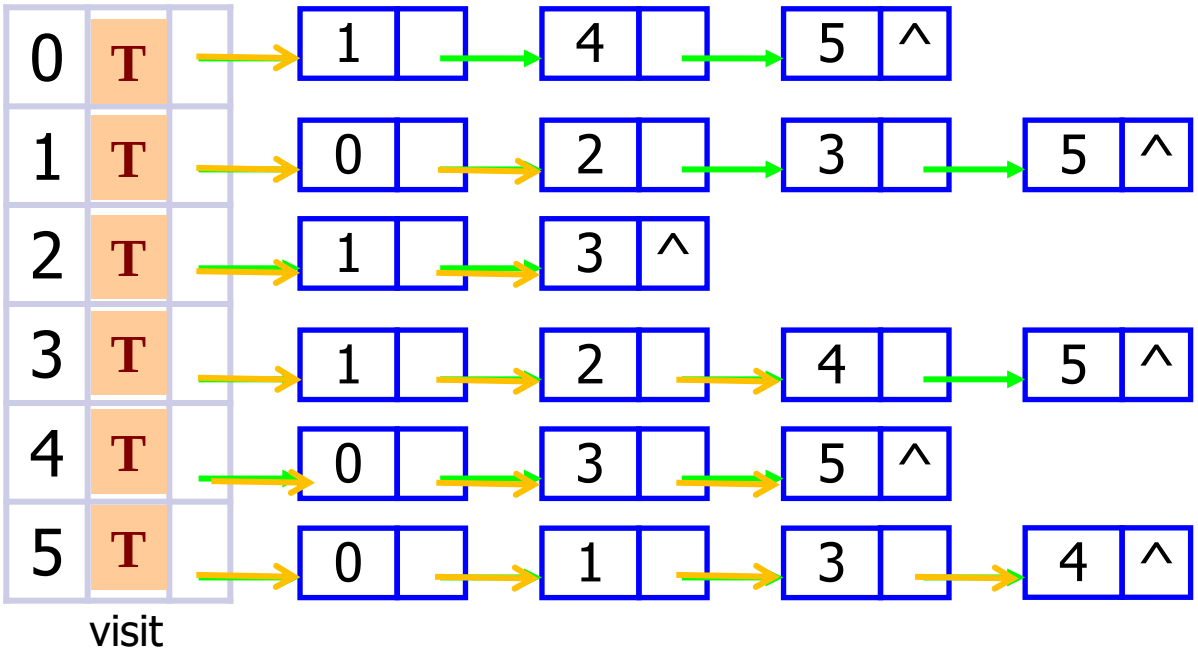
存储结构未定，则遍历顺序不确定

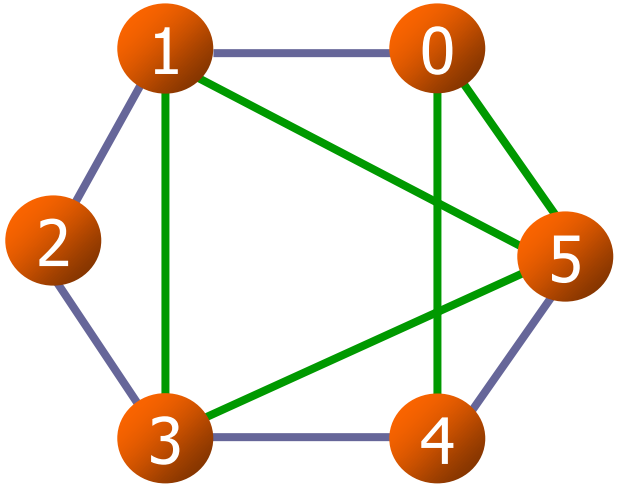
	0	1	2	3	4	5	6	7	8
访问标志：	T	T	T	T	T	T	T	T	T
	a	b	c	d	e	f	g	h	k
访问次序：	a	c	h	d	f	k	e	b	g

第三节　图的遍历

二、深度优先搜索(举例)

- 采用以下链表存储结构时，DFS次序为 0 1 2 3 4 5

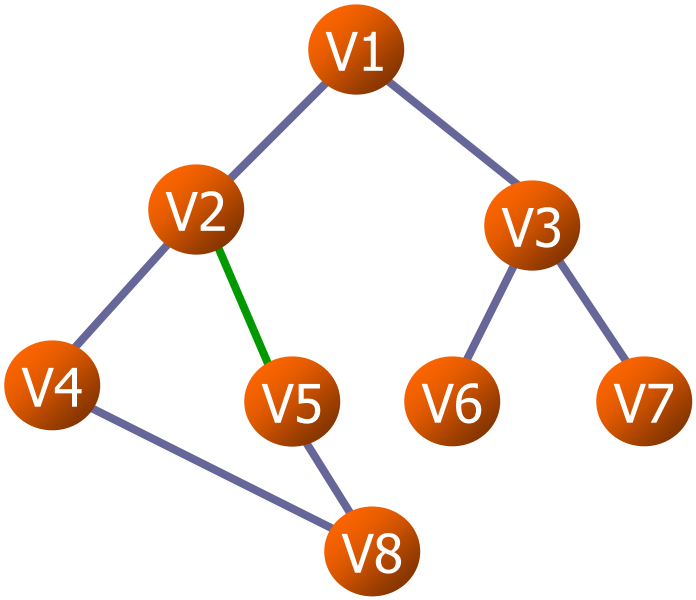




第三节　图的遍历

二、深度优先搜索(举例)

- DFS次序为 V1, V2, V4, V8, V5, V3, V6, V7



第三节　图的遍历

三、广度优先搜索(BFS)

- 广度优先搜索(BFS)是一种分层搜索方法
- BFS每向前走一步可能访问一批顶点，不存在往回退的情况
- BFS不是一个递归的过程。

第三节　图的遍历

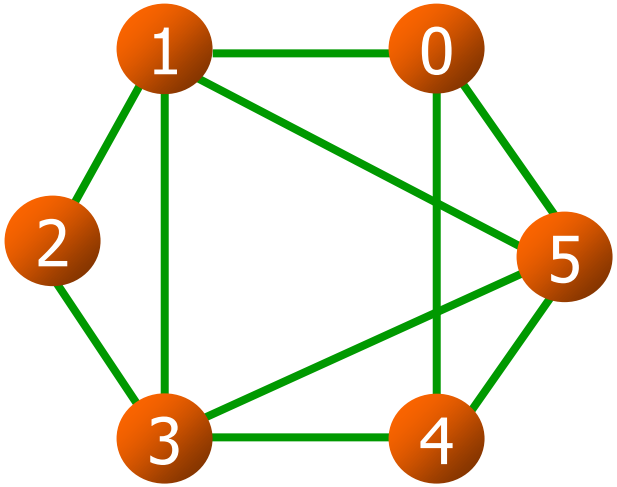
三、广度优先搜索(BFS算法)

- 所有顶点访问标志visited[]设置为FALSE
- 从某顶点v_0开始，访问v_0，visited[v_0]=TRUE，将v_0插入队列Q

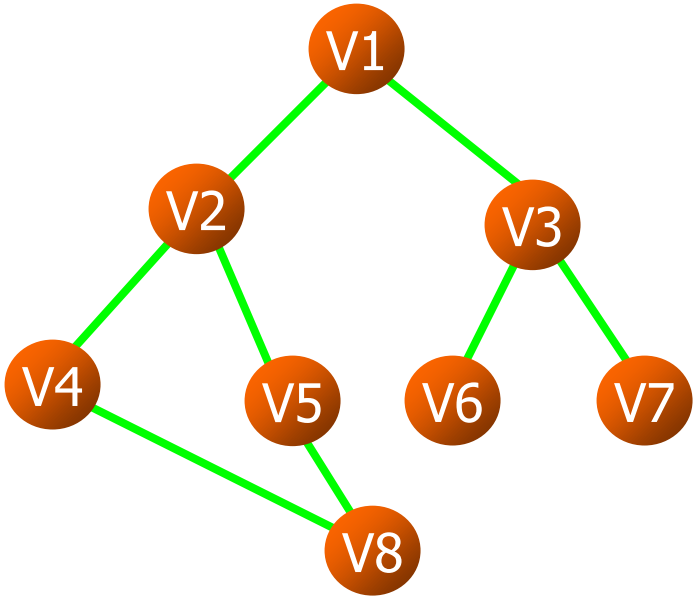
1. 如果队列Q不空，则从队列Q头上取出一个顶点v，否则结束
2. 依次找到顶点v的所有相邻顶点v’，如果visited[v’]=FALSE，访问该顶点v’，visited[v’]=TRUE，将v’插入队列Q
3. 重复1，2

第三节　图的遍历

三、广度优先搜索(举例)



BFS为0, 1, 4, 5, 2, 3



BFS为v1, v2, v3, v4,
v5, v6, v7, v8

结论

- 如果图为连通图，则从该图的任意一个顶点开始执行一次深度优先遍历或广度优先遍历，即可访问该连通图的所有顶点。
- 如果图为非连通图，则依次从未访问过的顶点开始执行深度优先遍历或广度优先遍历，直至所有的顶点均被访问。
- 事实上执行一次深度优先可以遍历一个连通分支。图有多少个连通分支，就调用多少次深度优先遍历。

- 深度优先搜索实现

```
void ALGraph::DFS(VexType v,bool visited[])
  //利用栈或递归实现从顶点v出发深度搜索图G

void ALGraph::DFSTraverse()
  //深度优先搜索图
  //增加辅助数组vistied记录图顶点是否访问
  //依某种顺序查找未被搜索的顶点v，调用DFS从v
  //出发深度优先搜索
```

思考：分析上述深度遍历的时间复杂度。

- 广度优先搜索实现

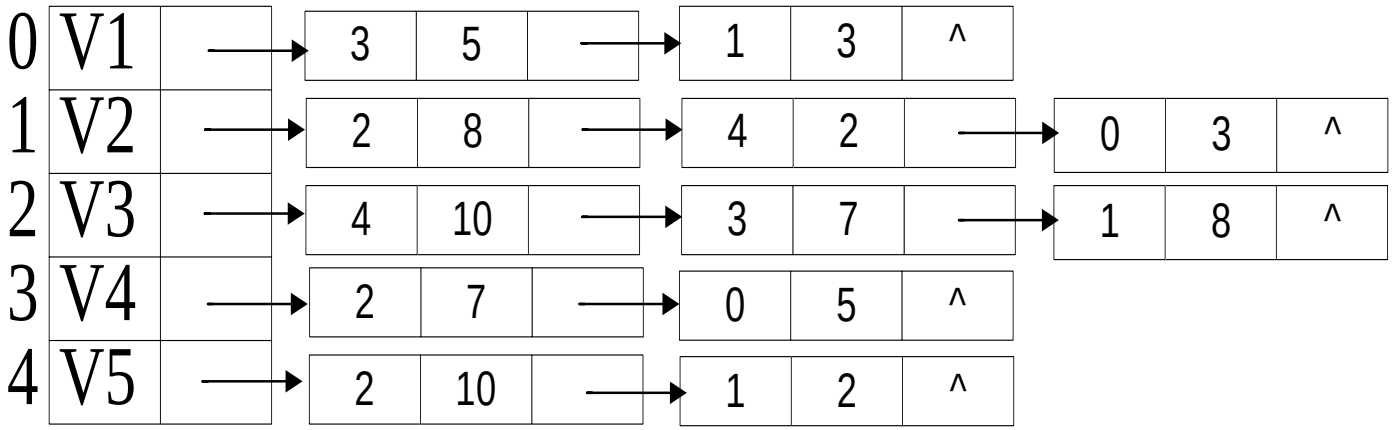
```
void ALGraph::BFS(VexType v,bool visited[])
  //利用队列实现从顶点v出发广度搜索图G

void ALGraph::BFSTraverse()
  //广度优先搜索图G
  //增加辅助数组vistied记录图顶点是否访问
  //依某种顺序查找未被搜索的顶点v，调用BFS从v
  //出发广度优先搜索
```

思考：分析上述广度遍历的时间复杂度。

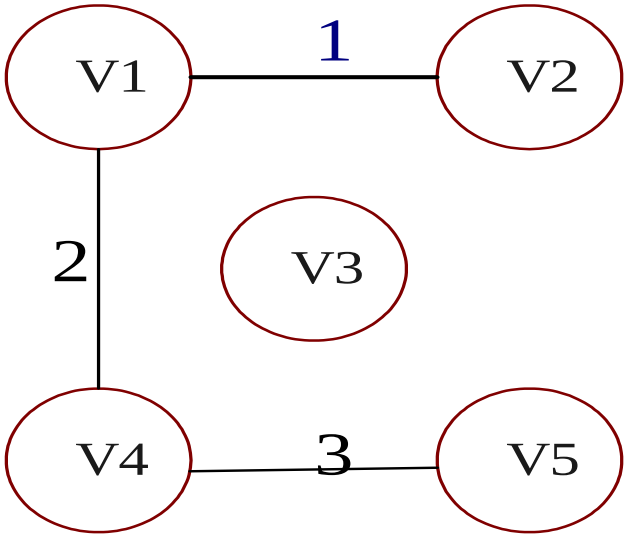
时间复杂度

- 可以看出无论是深度优先遍历还是广度优先遍历，其实质都是透过边或弧找邻接点的过程，只是访问的顺序不同。
- 两者的时间复杂度相同
- 取决于采取的存储结构，若用邻接矩阵为$O(N^2)$，若用邻接表则为$O(N+E)$

作业：假设无向网G的邻接表表示如下图，写出深度、广度优先遍历结果。



练习：假设用邻接表存储，下图中边上序号表示边输入顺序(链表头插入)，画出该图邻接表，写出用该邻接表存储时其深度优先顺序和广度优先顺序。



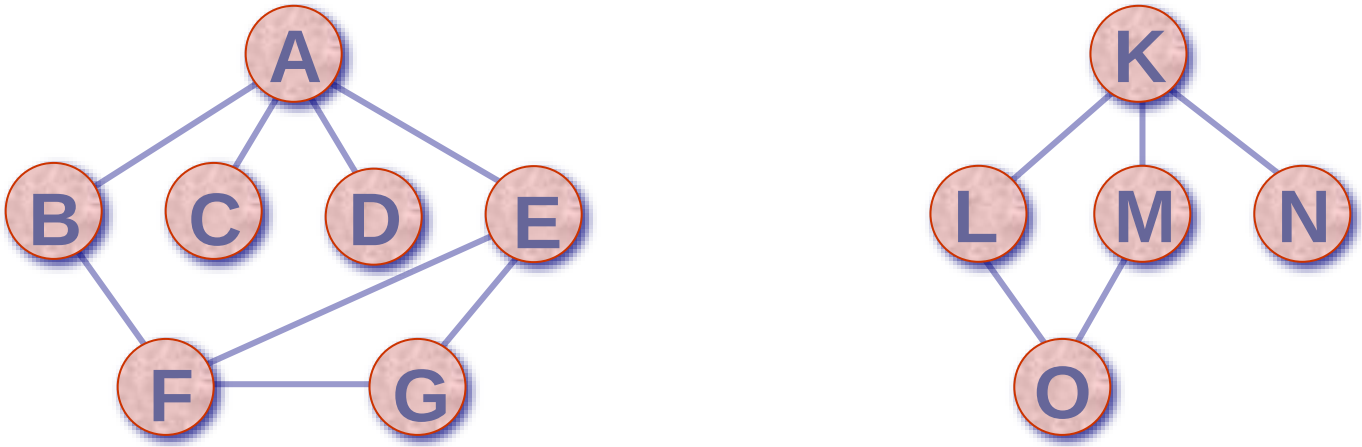
无向图G5

7.4 图的连通性问题

第四节　图的连通性问题

一、无向图的连通性

- 如果无向图中，存在不连通的顶点，则该图称为非连通图



第四节　图的连通性问题

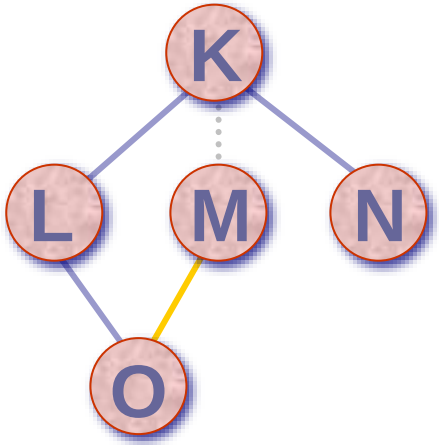
二、无向图的连通分量

- 非连通图的极大连通子图叫做连通分量
- 若从无向图的每一个连通分量中的一个顶点出发进行DFS或BFS遍历，可求得无向图的所有连通分量的生成树(DFS或BFS生成树)
- 所有连通分量的生成树组成了非连通图的生成森林

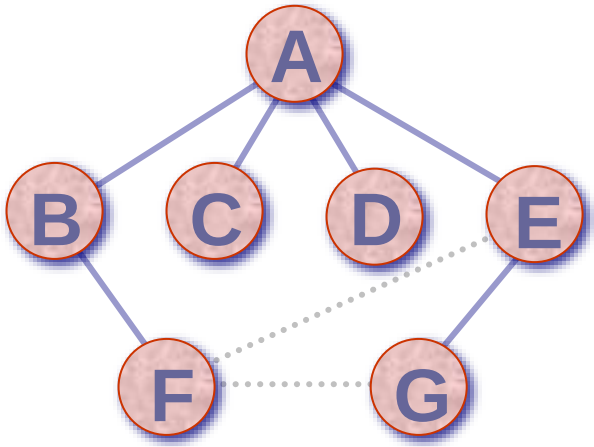
第四节　图的连通性问题

二、无向图的生成树

- 由DFS遍历，求得连通分量称为DFS生成树
- 由BFS遍历，求得连通分量称为BFS生成树



DFS生成树



BFS生成树

第四节　图的连通性问题

三、最小生成树

- 如果无向图中，边上有权值，则称该无向图为无向网
- 如果无向网中的每个顶点都相通，称为连通网
- 最小生成树(Minimum Cost Spanning Tree)是代价最小的连通网的生成树，即该生成树上的边的权值和最小

第四节　图的连通性问题

三、最小生成树(准则)

- 必须使用且仅使用连通网中的n-1条边来联结网络中的n个顶点；
- 不能使用产生回路的边；
- 各边上的权值的总和达到最小。
- 常用于道路建设、线路铺设等应用中计算成本。

第四节　图的连通性问题

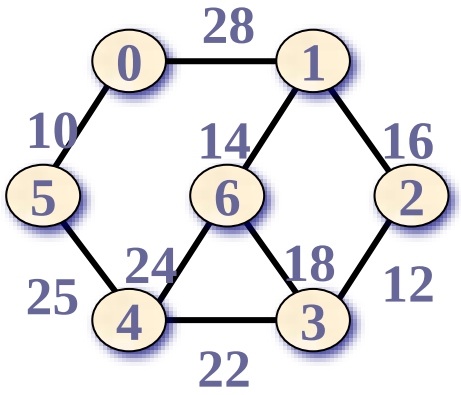
四、普里姆(Prim)算法生成最小生成树

- 假设N=(V, E)是连通网
- TE是N上最小生成树中边的集合

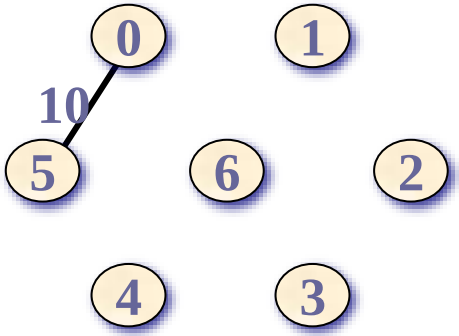
1. U={u_0}，（$u_0 \in V$），TE={}
2. 在所有$u \in U, v \in V-U$的边$(u, v) \in E$中找一条代价最小的边(u, v_0)并入集合TE，同时v_0并入U
3. 重复2，直到U=V。T=(V，TE)即为所求最小生成树。

第四节　图的连通性问题

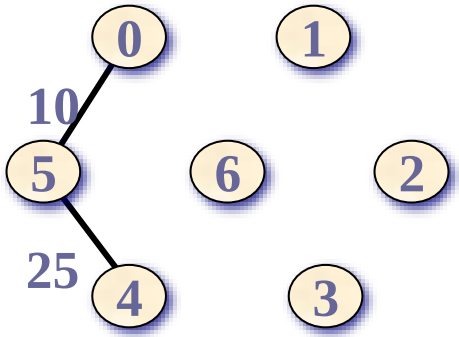
四、普里姆(Prim)算法举例



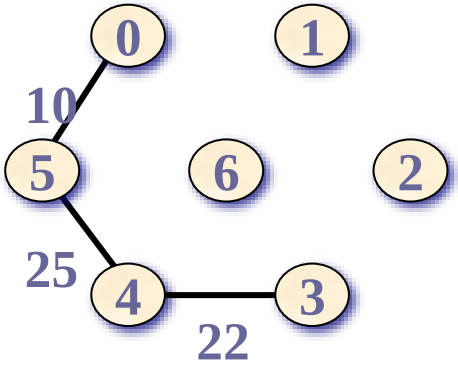
原图



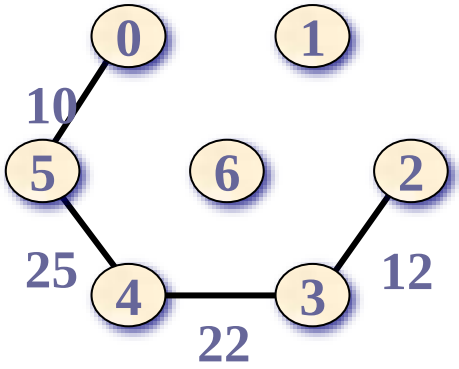
(a)



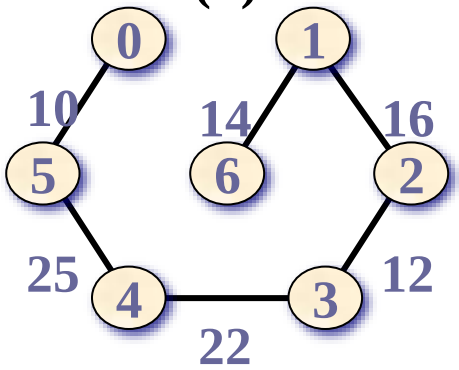
(b)



(c)



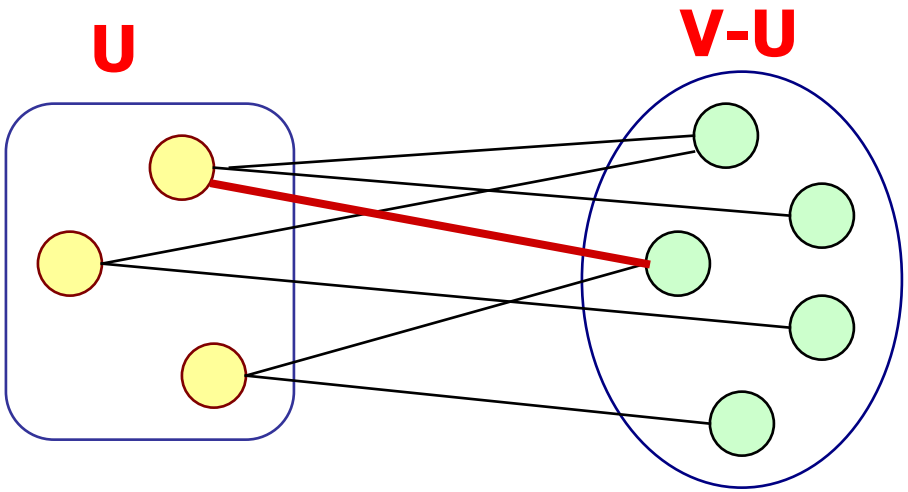
(d)



(e) (f)

第四节　图的连通性问题

在生成树的构造过程中，图中 n 个顶点分属两个集合：已落在生成树上的顶点集 U 和尚未落在生成树上的顶点集V-U，应在所有连通U中顶点和V-U中顶点的边中选取权值最小的边逐渐加入TE，相应顶点加入U中。



Prim算法实现

为直观化，以表格来表示实现过程中各变量变化和节点加入情况。

其中变量定义如下：

visited —— 数组，表示顶点v是否加入生成树U中，加入，为1，否则，0。

dis —— 数组，表示从U到V-U中各顶点的最小权值。

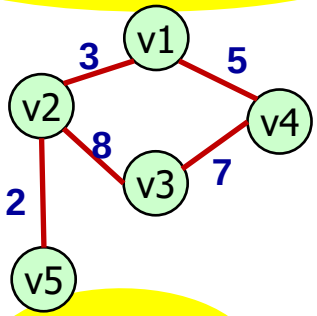
adj —— 数组，使dis取最小的U中邻接点。

mindis —— dis中的最小值

flag —— mindis取最小对应的顶点v

V1	V2	V3	V4	V5	misdis	flag	U
0 ∞	0 ∞	0 ∞	0 ∞	0 ∞		V1	V1
1	0 3 V1	0 ∞	0 5 V1	0 ∞	3	V2	V1,V2
1	1 3 V1	0 8 V2	0 5 V1	0 2 V2	2	V5	V1,V2,V5
1	1 3 V1	0 8 V2	0 5 V1	1 2 V2	5	V4	V1,V2 V5,V4
1	1 3 V1	0 7 V4	1 5 V1	1 2 V2	7	V3	V1,V2,V5, V4,V3

程序中不需要



初始化

第1行：vistied
第2行：dis
第3行：adj

选取同行中visited为0，dis最小的值

Adj数组中存储的的是结点信息or结点下标？

生成树生成中表格变化。

```
class ALGraph
{
    private:
        VNode*vertices;
        int             ArcNum;
        int             VexNum;
        int             GKind;
        int             GetLocVex(char  vex[]);
        void            BFS(char vex[],bool visited[]);
        void            DFS(char vex[],bool visited[]);

    public:
        ALGraph(){};
        void CreateALGraph();
        void BFSTraverse( );
        void DFTraverse( );
        void Prim( );
};
```

void ALGraph::Prim()

```
//实现prim算法，输出树的各边
{
    依据某顶点(下标为0)初始化visited,adj,dis;
    for(i=0; i<vexnum-1; i++) {
        misdis=visited等于0的各顶点中dis最小值；
        flag=取misdis的顶点；
        输出边adj[flag],flag;
        visited[flag]=1;
        修改flag顶点到未加入树的各顶点的dis；
     }
}
```

第四节　图的连通性问题

五、克鲁斯卡尔(Kruskal)算法生成最小生成树

- 假设N=(V, E)是连通网

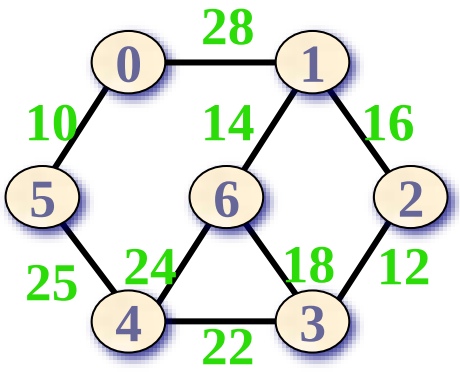
1. 非连通图T={V, {}}，图中每个顶点自成一个连通分量

2. 在E中找一条代价最小，且其两个顶点分别依附不同的连通分量的边，将其加入T中

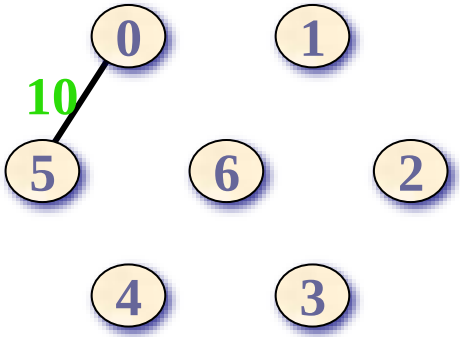
3. 重复2，直到T中所有顶点都在同一连通分量上

第四节　图的连通性问题

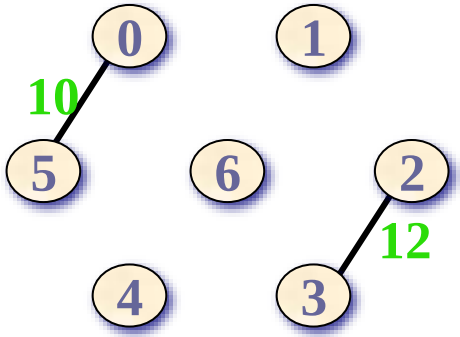
五、克鲁斯卡尔(Kruskal)算法举例



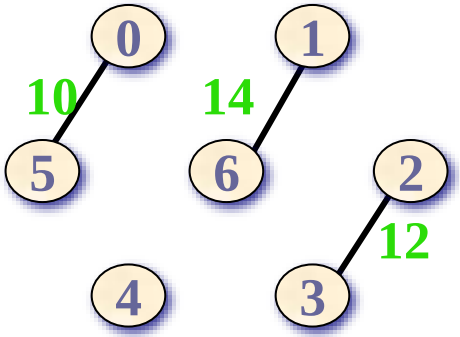
原图



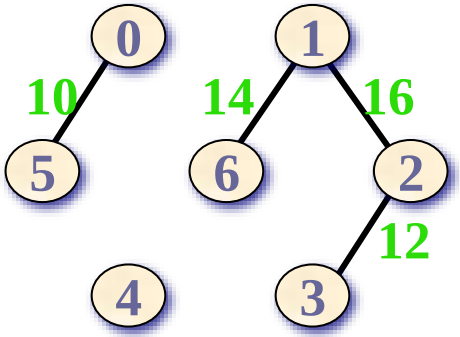
(a)



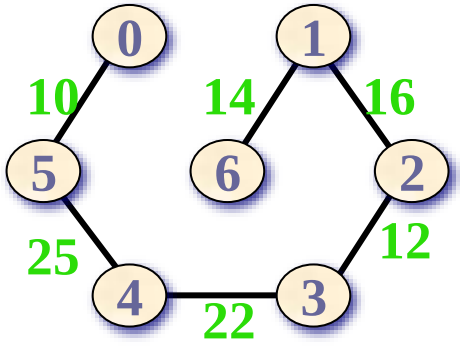
(b)



(c)



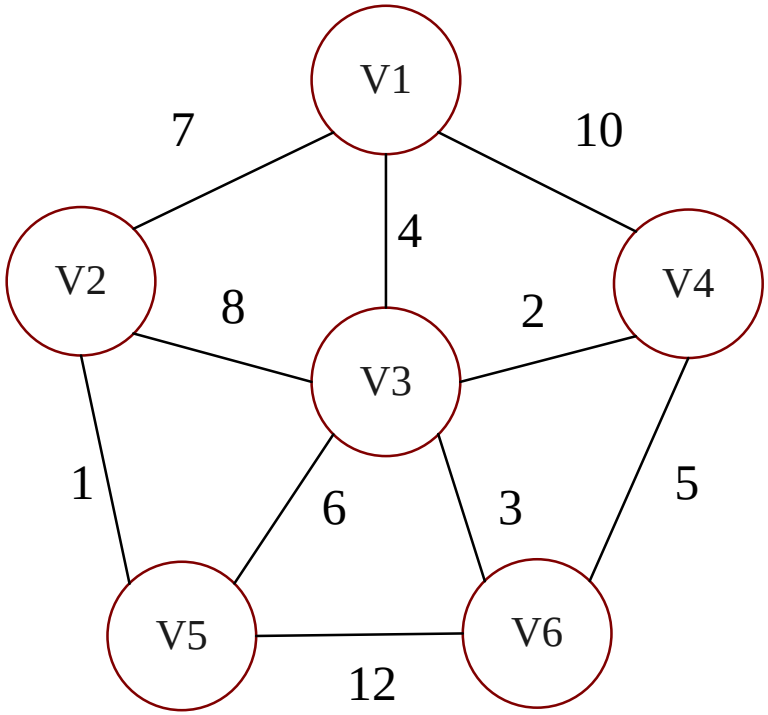
(d)



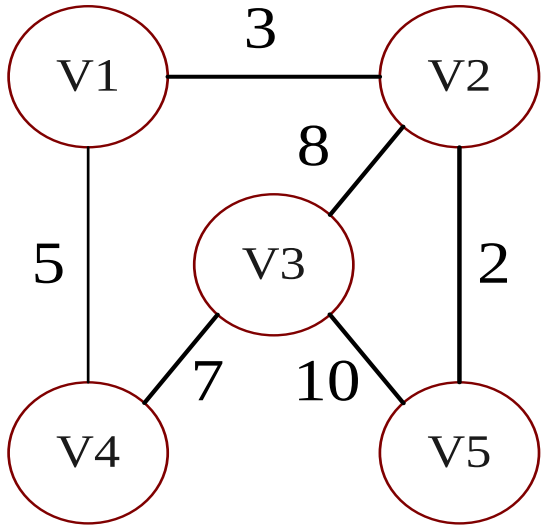
(e) (f)

练习：用Prim算法求下图的最小生成树，给出生成过程，画出计算表格

给出克鲁斯卡尔算法的生成过程



作业：用Prim和Kruskal算法分别求下图的最小生成树，给出生成过程。



```
class ALGraph {
  private:
      VNode       *vertices;
      int         ArcNum;
      int         VexNum;
      int         GKind;
      int         GetLocVex(char  vex[]);
      void        BFS(char vex[],bool visited[]);
      void        DFS(char vex[],bool visited[]);
  public:
      ALGraph(){};
      void CreateALGraph();
      void BFSTraverse( );
      void DFTraverse( );
      void Prim(char vex[]);
   void Kruskal();    };
```

克鲁斯卡尔算法需考虑的问题

目标

- 按照从小到大的顺序尝试每条边
- 判断边的顶点不在同一个联通分支
- 在生成树中加入这条边

方法

- 把边按照从小到大的顺序排序
- 为不同的联通分支编号，可为顶点加一个属性叫做联通分支编号
- 修改生成树中的顶点为同一个编号

比较两种算法

算法名	普里姆算法	克鲁斯卡尔算法
时间复杂度	$O(n^2)$	O(eloge)
适应范围	稠密图	稀疏图

7.5 最短路径

第五节　最短路径

思考：

（1）从图中一个顶点到另外顶点，如何使途中经过的顶点最少？　（广度优先）

（2）从图中一个点出发，如何到其它点的路径最短？

（单源点最短路径，本节内容）

第五节　最短路径

一、最短路径

- 最短路径是求从图（或网）中某一顶点，到其余各顶点的最短路径
- 最短路径与最小生成树主要有三点不同：

1. 最短路径的操作对象主要是有向图(网)，而最小生成树的操作对象是无向图
2. 最短路径有一个始点，最小生成树没有
3. 最短路径关心的是始点到每个顶点的路径最短，而最小生成树关心的是整个树的代价最小

第五节　最短路径

二、Dijkstra算法

- 最短路径可以采用迪杰斯特拉(Dijkstra)算法求解
- Dijkstra算法采用按路径长度递增的次序产生最短路径

第五节　最短路径

基本概念

路径长度:一条路径上所经过的边的数目

带权路径长度:路径上所经过边的权值之和

最短路径:(带权)路径长度(值)最小的那条路径

最短路径长度或最短距离:最短路径长度

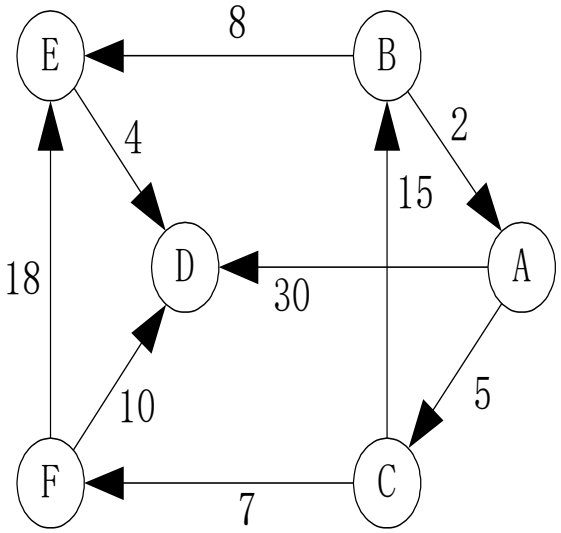
迪杰斯特拉(Dijkstra)算法

迪杰斯特拉算法的思想是：按路径长度递增的顺序逐步产生最短路径，

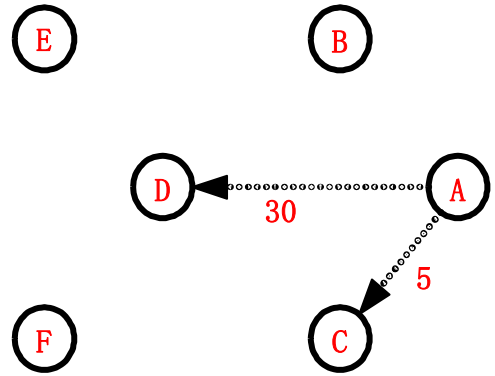
1. 设置两个顶点的集合U和T，集合U中存放已找到最短路径的顶点，集合T中存放当前还未找到最短路径的顶点。
2. 初始状态时，集合U中只包含源点，设为v_0;
3. 然后从集合T中选择到源点v_0路径长度最短的顶点u加入到集合U中；
4. 集合U中每加入一个新的顶点u都要修改源点v_0到集合T中剩余顶点的当前最短路径长度值，集合T中各顶点的新的当前最短路径长度值，为原来的当前最短路径长度值与从源点过顶点u到达该顶点的路径长度中的较小者。
5. 转到3，此过程不断重复，直到集合T中的顶点全部加入到集合U中为止。

第五节　最短路径

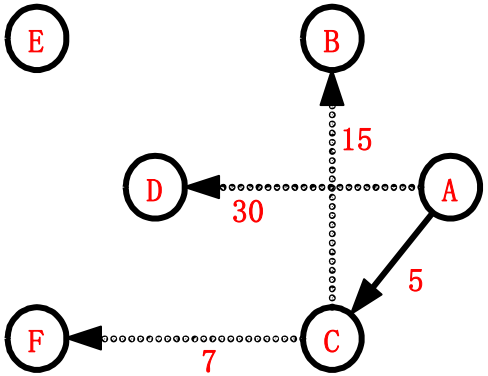
例：求下图A顶点到各顶点的最短路径。



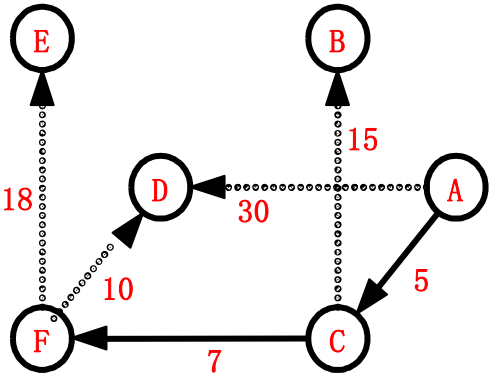
$$\begin{bmatrix} \infty & \infty & 5 & 30 & \infty & \infty \\ 2 & \infty & \infty & \infty & 8 & \infty \\ \infty & 15 & \infty & \infty & \infty & 7 \\ \infty & \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & 4 & \infty & \infty \\ \infty & \infty & \infty & 10 & 18 & \infty \end{bmatrix}$$



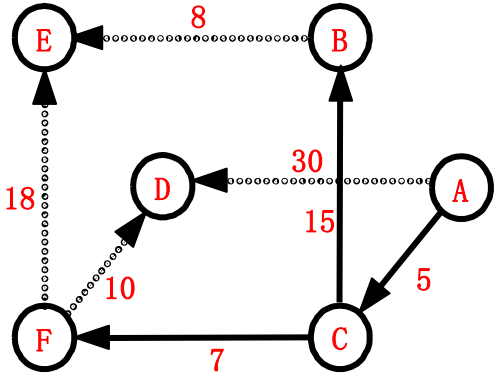
(a)



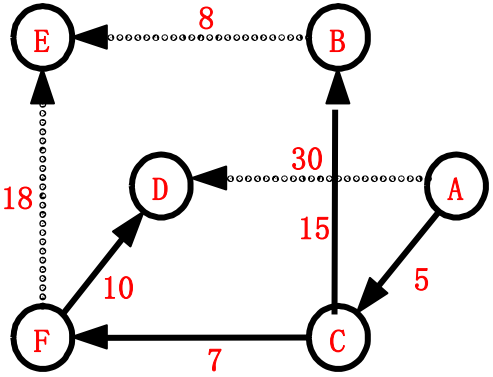
(b)



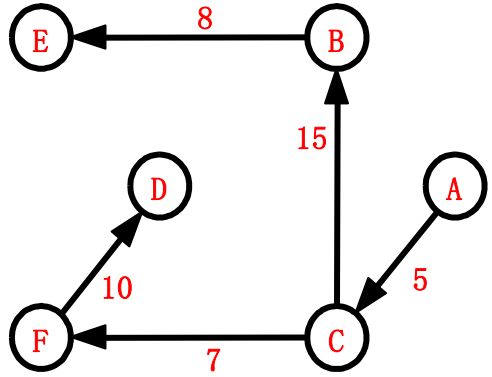
(c)



(d)



(e)



(f)

迪杰斯特拉算法求从顶点A到其余各顶点最短路径的过程

第五节　最短路径

二、Dijkstra算法

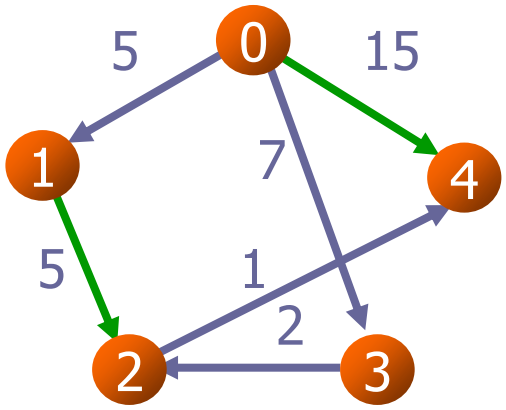
- 在Dijkstra算法中，引进了一个辅助向量D
- 每个分量D[i]表示当前所找到的从始点到每个终点v_i的最短路径长度
- D[i]初值为始点v_0到各终点v_i的直接距离，即若从始点到某终点有(出)弧，则为弧上的权值，否则为∞

第五节　最短路径

二、Dijkstra算法

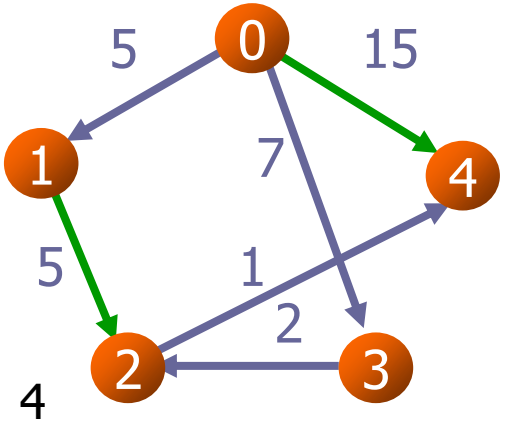
- 如何得出从开始点到各个顶点的最短路径？（经过哪些顶点）
- 设置另一个辅助向量path[]，用来存放得到的从源点v0到其余各顶点的最短路径上到达目标顶点的前一顶点下标。

	0	1	2	3	4
path	-1	0	3	0	2



最短路径的路径保存方法二

- 为每一个顶点i设置辅助向量path[i][]，用来存放得到的从源点v0到该顶点的最短路径中依次访问过的顶点。
- 第一个值是路径上的顶点数。



	0	1	2	3	4
path[0][]	2	0	0	0	0
path[1][]	2	0	1	0	0
path[2][]	3	0	3	2	0
path[3][]	2	0	3	0	0
path[4][]	4	0	3	2	4

```
class ALGraph {
    private:
        VexNode      *vertices;
        int          ArcNum;
        int          VexNum;
        int          GKind;
        int          GetLocVex(char  vex[]);
        void         BFS(char vex[],bool visited[]);
        void         DFS(char vex[],bool visited[]);
    public:
        ALGraph(){};
        void CreateALGraph();
        void BFSTraverse( );
        void DFTraverse( );
        void Prim(char vex[]);
     void Kruskal();
     void Dijkstra(char var[])
     //从顶点var出发到各顶点的最短路径及路径值,
};
```

迪杰斯特拉算法描述：

(1) 初始：final[0]=1(U={V_0}) final[i]=0(T=V-U)
D[i]=∞。修改从v0出发的邻接点vi的D值和path
D[i]= c_{0i} path[i]={2,0,i};

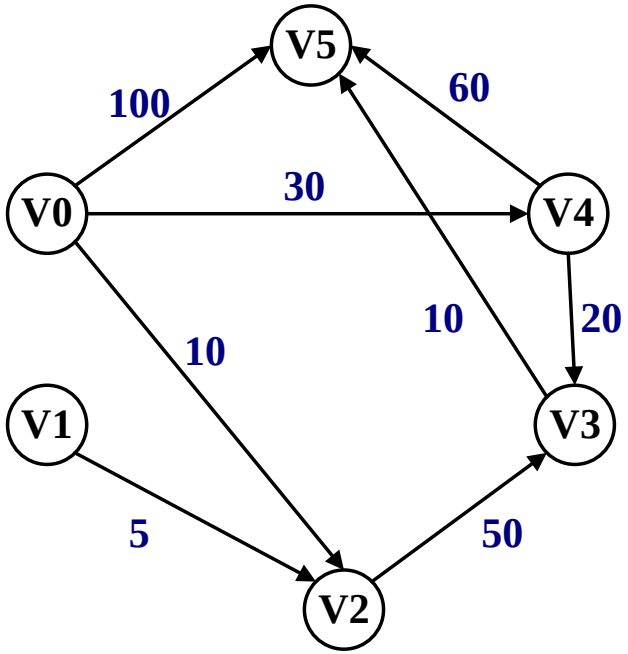
(2) 求最短路径，令：D[j] = min{D[i]|final[i]=0}

(3) 输出：v_j: path[j][1]---path[j][path[j][0]],
D[j]
final[j]=1(U=U+{v_j})

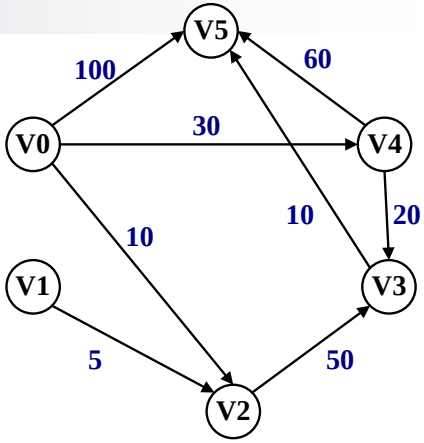
(4) 修改v_j的邻接点v_k的D值和path
若final(k)=0且D[j]+c_{jk}<D[k],则
D[k] = D[j]+c_{jk}
path[k] = path[j] （复制先到j的路径）
path[k][0]++; path[k][path[k][0]]=k;

(5) 重复（2）直到U=V或无最小值。

练习5：对下图求从V_0出发到各顶点的最短路径。



求解过程见下表。其中每一列代表一个选择D、修改D的过程。单元格内的第一行是final、第二行是D值、第三行是path值。



终点	从v_0到各终点的D值和最短路径的求解过程				
	i=1	i=2	i=3	i=4	i=5
v_1	0 ∞	0 ∞	0 ∞	0 ∞	0 ∞ 无
v_2	0 10 $\{v_0,v_2\}$	1			
v_3	0 ∞	0 60 $\{v_0,v_2,v_3\}$	0 50 $\{v_0,v_4,v_3\}$	1	
v_4	0 30 $\{v_0,v_4\}$	0 30 $\{v_0,v_4\}$	1		
v_5	0 100 $\{v_0,v_5\}$	0 100 $\{v_0,v_5\}$	0 90 $\{v_0,v_4,v_5\}$	0 60 $\{v_0,v_4,v_3,v_5\}$	
v_j	v_2	v_4	v_3	v_5	
U	$\{v_0,v_2\}$	$\{v_0,v_2,v_4\}$	$\{v_0,v_2,v_4,v_3\}$	$\{v_0,v_2,v_4,v_3,v_5\}$	

作业：

有向网N={V,E}，V={0,1,2,3,4}，E={<0,1,1>，<0,3,3>，<0,4,10>，<1,2,5>，<2,4,1>，<3,2,2>，<3,4,6>}，E中每个元组的第三个元素表示权。

1、画出该网。

2、写出该网的邻接矩阵。

3、用Dijkstra算法求最短路径，写出顶点0到其它各顶点的最短路径长度、路径及产生过程。

求n个顶点之间的最短路径

- 用Dijkstra算法也可以求得有向图G=(V，E)中每一对顶点间的最短路径。
- 方法是：设置二维数组D[i][j]，数组每一行D[i]表示从顶点vi出发到其它顶点的最短路径，即每次以一个不同的顶点v_i为源点重复Dijkstra算法便可求得每一对顶点间的最短路径，时间复杂度是$O(n^3)$。

求n个顶点之间的最短路径

- 弗罗伊德(Floyd)算法，其时间复杂度仍是$O(n^3)$，但算法形式更为简明，步骤更为简单，数据结构是基于图的邻接矩阵。

弗罗伊德算法思想

- 设顶点集S(初值为空)，用数组A的每个元素A[i][j]保存从V_i只经过S中的顶点到达V_j的最短路径长度：

① 初始时令S={ }， A[i][j]的赋初值方式是：

$$A[i][j]=\begin{cases} 0 & i=j\text{时} \\ W_{ij} & i\neq j\text{且}<v_i,v_j>\in E，\ w_{ij}\text{为弧上的权值} \\ \infty & i\neq j\text{且}<v_i,v_j>\text{不属于}E \end{cases}$$

② 将图中一个顶点V_k 加入到S中，修改A[i][j]的值，修改方法是：

A[i][j] = Min{ A[i][j] , (A[i][k]+A[k][j]) }

原因： 从V_j只经过S中的顶点(V_k)到达V_j的路径长度可能比原来不经过V_k的路径更短。

③ 重复②，直到G的所有顶点都加入到S中为止。

弗罗伊德算法实现(1)

弗罗伊德算法实现

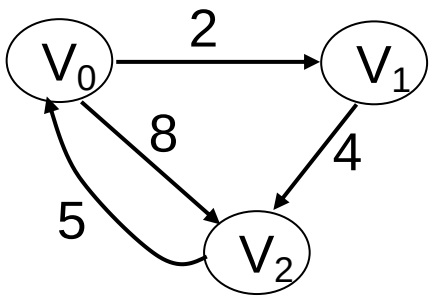
- 定义二维数组Path[n][n](n为图的顶点数)，元素Path[i][j]保存从V_i到V_j的最短路径所经过的顶点。

  若Path[i][j]=k：从V_i到V_j经过V_k，最短路径序列是(V_i，…, V_k，…, V_j)，则路径序列：(V_i，…, V_k)和(V_k，…, V_j)一定是从V_i到V_k和从V_k到V_j的最短路径。从而可以根据Path[i][k]和Path[k][j]的值再找到该路径上所经过的其它顶点，…依此类推。

弗罗伊德算法实现(2)

- **弗罗伊德算法实现**

初始时令Path[i][j]=-1，表示从V_i到V_j不经过任何(S中的中间)顶点。当某个顶点V_k加入到S中后使A[i][j]变小时，令Path[i][j]=k。



带权有向图及其邻接矩阵

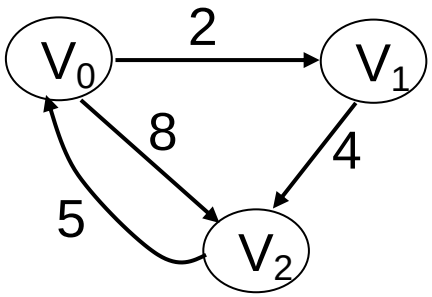
$$\begin{bmatrix} 0 & 2 & 8 \\ \infty & 0 & 4 \\ 5 & \infty & 0 \end{bmatrix}$$

$$\begin{bmatrix} -1 & 1 & 2 \\ -1 & -1 & 2 \\ 0 & -1 & -1 \end{bmatrix}$$

Path[ ][ ]

弗罗伊德算法实现举例

步骤	初始	k=0	K=1	K=2
A	$\begin{bmatrix} 0 & 2 & 8 \\ \infty & 0 & 4 \\ 5 & \infty & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 2 & 8 \\ \infty & 0 & 4 \\ 5 & 7 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 2 & 6 \\ \infty & 0 & 4 \\ 5 & 7 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 2 & 6 \\ 9 & 0 & 4 \\ 5 & 7 & 0 \end{bmatrix}$
Path	$\begin{bmatrix} -1 & 1 & 2 \\ -1 & -1 & 2 \\ 0 & -1 & -1 \end{bmatrix}$	$\begin{bmatrix} -1 & 1 & 2 \\ -1 & -1 & 2 \\ 0 & 0 & -1 \end{bmatrix}$	$\begin{bmatrix} -1 & 1 & 1 \\ -1 & -1 & 2 \\ 0 & 0 & -1 \end{bmatrix}$	$\begin{bmatrix} -1 & 1 & 1 \\ 2 & -1 & 2 \\ 0 & 0 & -1 \end{bmatrix}$
S	{ }	{ 0 }	{ 0, 1 }	{ 0, 1, 2 }



根据上述过程中Path[i][j]数组，得出：

V_0到V_1：最短路径是{ 0, 1 }，路径长度是2；

V_0到V_2：最短路径是{ 0, 1, 2 }，路径长度是6；

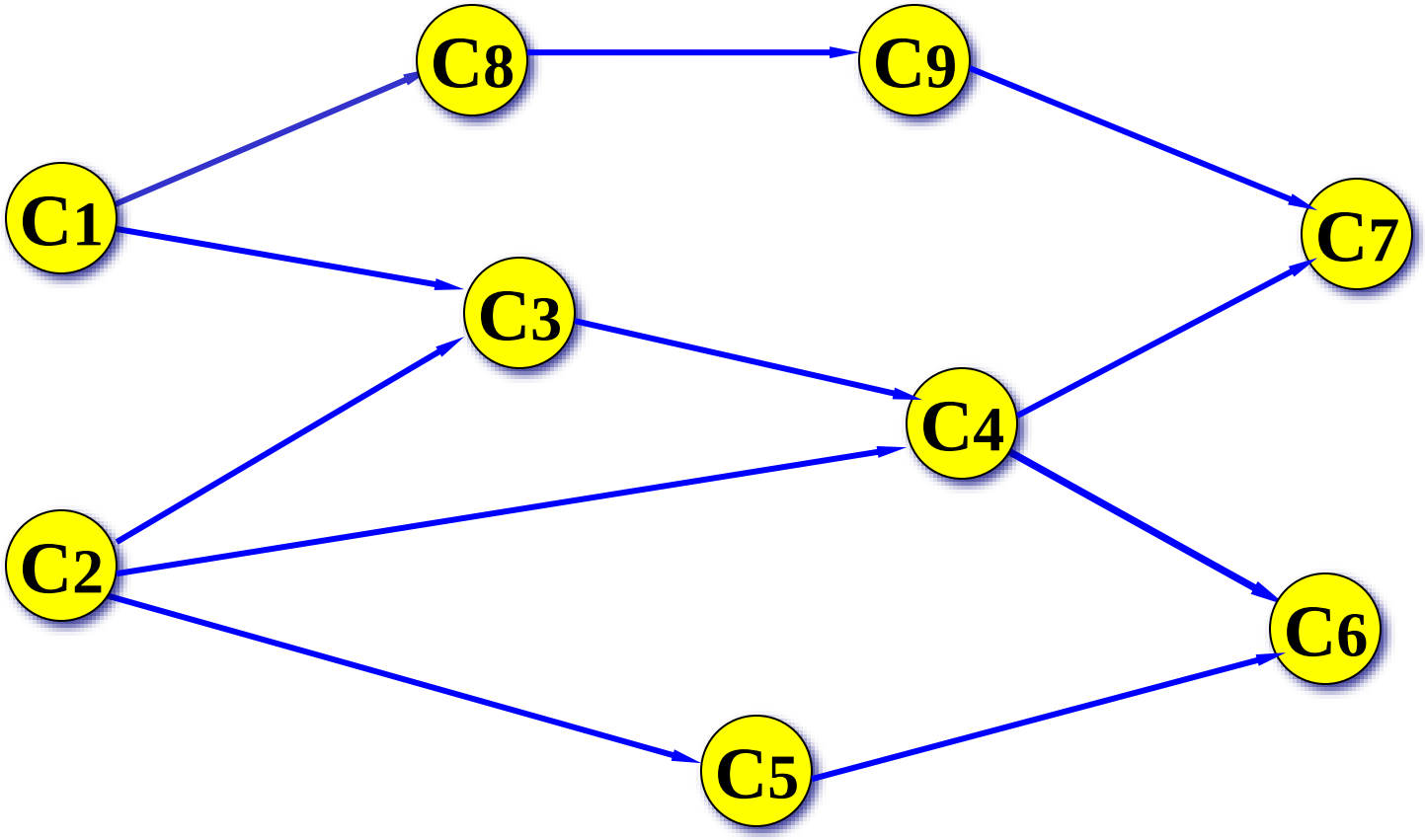
V_1到V_0：最短路径是{ 1, 2, 0 }，路径长度是9；

7.6 有向无环图及其应用

第六节　有向无环图及其应用

- 计划、施工过程、生产流程、程序流程等都是“工程”。除了很小的工程外，一般都把工程分为若干个叫做“活动”的子工程。完成了这些活动，这个工程就可以完成了。
- 计算机专业学生的学习就是一个工程，每一门课程的学习就是整个工程的一些活动。其中有些课程要求先修课程，有些则不要求。这样在有的课程之间有领先关系，有的课程可以并行地学习。

课程代号	课程名称	先修课程
C_1	高等数学	
C_2	程序设计基础	
C_3	离散数学	C_1, C_2
C_4	数据结构	C_3, C_2
C_5	高级语言程序设计	C_2
C_6	编译方法	C_5, C_4
C_7	操作系统	C_4, C_9
C_8	普通物理	C_1
C_9	计算机原理	C_8



学生课程学习工程图

第六节　有向无环图及其应用

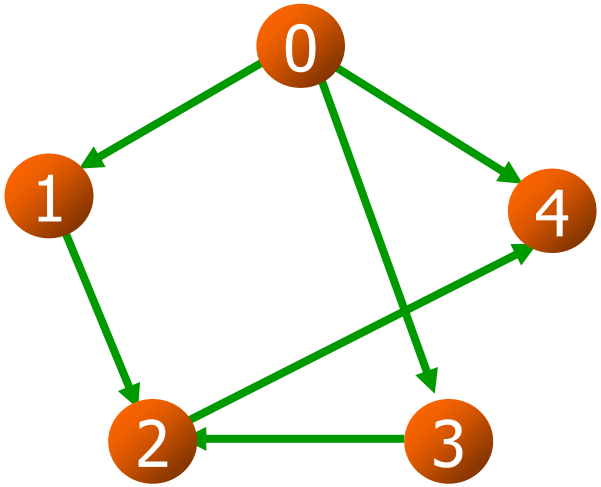
一、AOV-网

- 可以用有向图表示一个工程。在这种有向图中，用顶点表示活动，用有向边<V_i, V_j>表示活动V_i必须先于活动V_j进行。这种有向图叫做顶点表示活动的AOV网络(Activity On Vertices)。
- 在AOV网络中不能出现有向回路，即有向环。如果出现了有向环，则意味着某项活动应以自己作为先决条件。
- 因此，对给定的AOV网络，必须先判断它是否存在有向环。

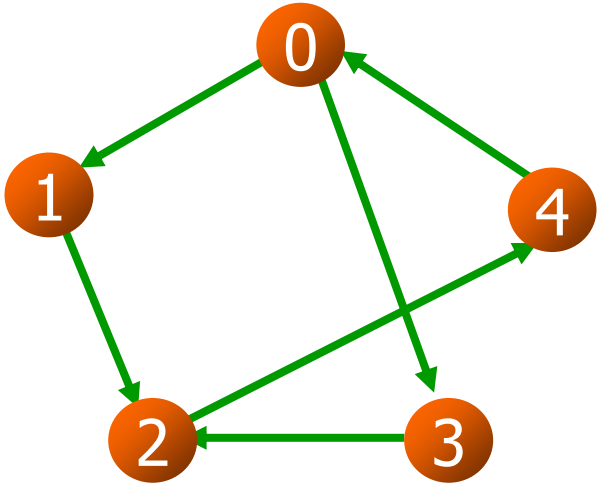
第六节　有向无环图及其应用

二、有向无环图(DAG)

- 有向无环图(DAG:Directed Acycline Graph)是图中无环的有向图



DAG



非DAG

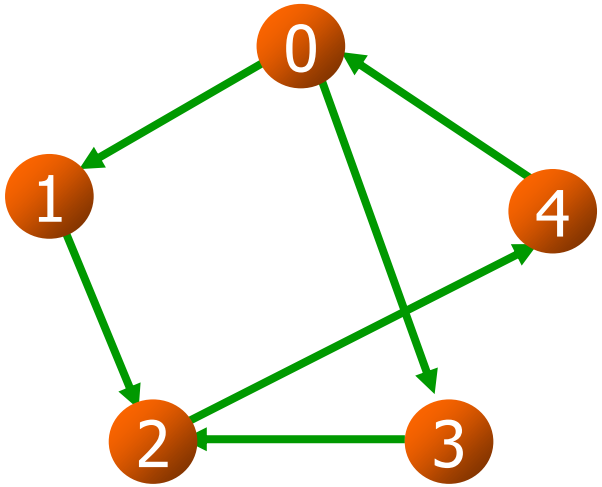
第六节　有向无环图及其应用

问题：　如何检查有向图中是否有回路呢？

解决方法：深度优先搜索

拓扑排序(本节)

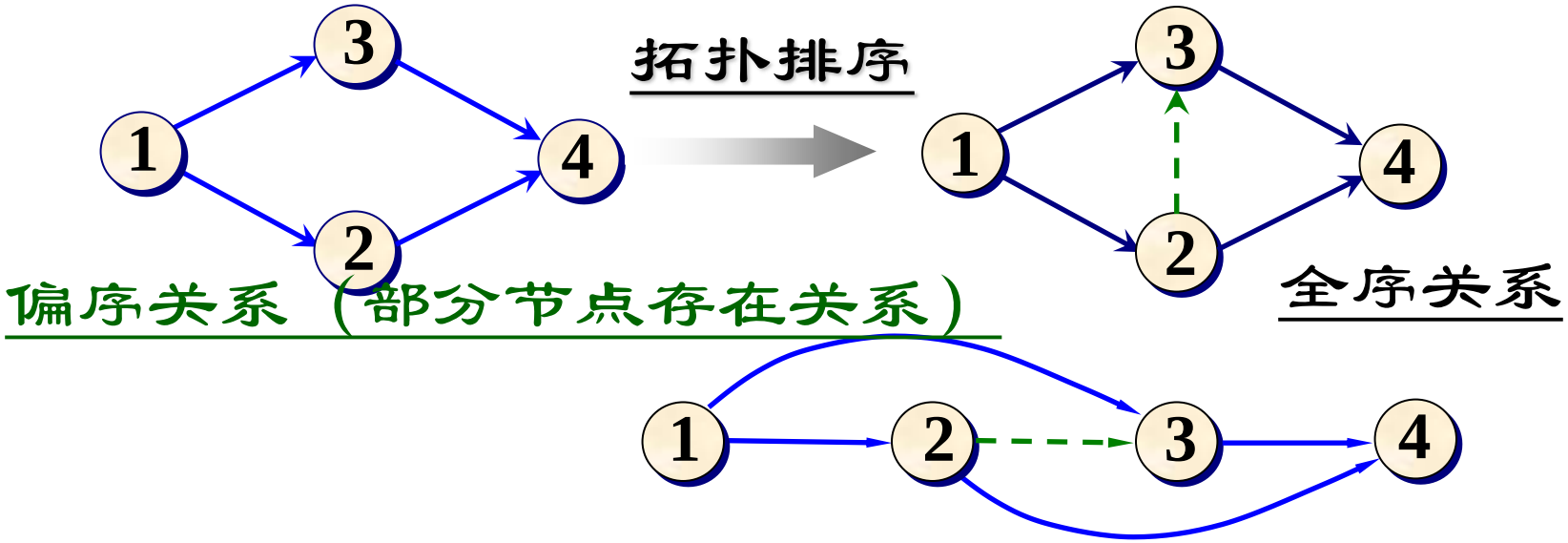
练习：用深度优先搜索(DFS)判定下图是否DAG图。



- 从某个顶点v出发，进行DFS，如果存在一条从顶点u到v的回边，则有向图中存在环。
- DFS：0, 1, 2, 4, 3

三、拓扑排序

- 检测有向环的一种方法是对AOV网络构造它的**拓扑有序序列**。即将各个顶点（代表各个活动）排列成一个线性有序的序列，使得AOV网络中所有应存在的前驱和后继关系都能得到满足。



第六节　有向无环图及其应用

三、拓扑排序

1. 拓扑排序

- 由严格偏序定义得到的拓扑有序的操作称拓扑排序
- 算法：

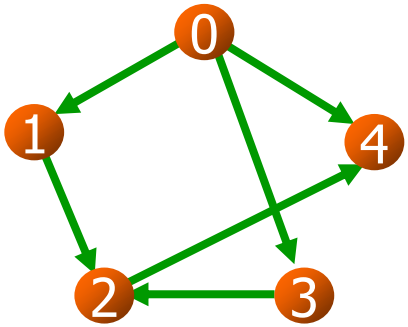
(1). 在有向图中选一个没有前驱的顶点且输出之

(2). 从图中删除该顶点和所有以它为尾的弧

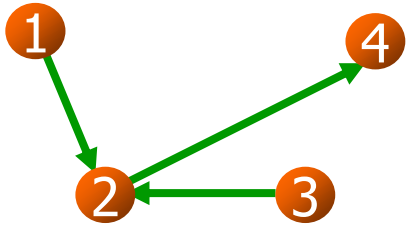
重复(1)(2)两步，直到所有顶点输出为止或跳出循环。

第六节　有向无环图及其应用

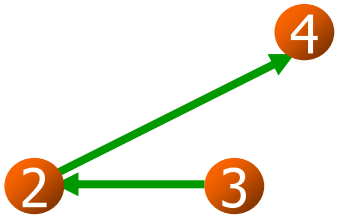
三、拓扑排序(举例)



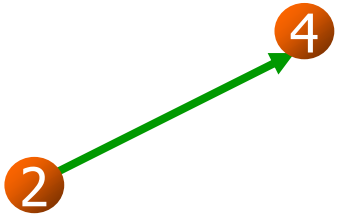
原图



输出0之后



输出0,1之后



输出0,1,3之后

输出0,1,3,2之后

最后输出拓扑排序结果：
0,1,3,2,4

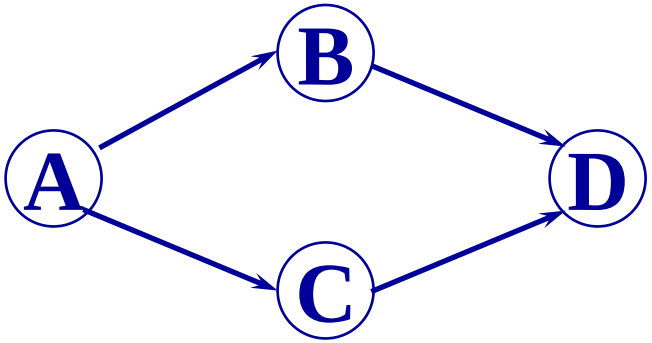
拓扑排序与AOV网

- 这种构造AOV网络全部顶点的拓扑有序序列的运算就叫做拓扑排序。

- 如果通过拓扑排序能将AOV网络的所有顶点都排入一个拓扑有序的序列中，则该网络中必定不会出现有向环。

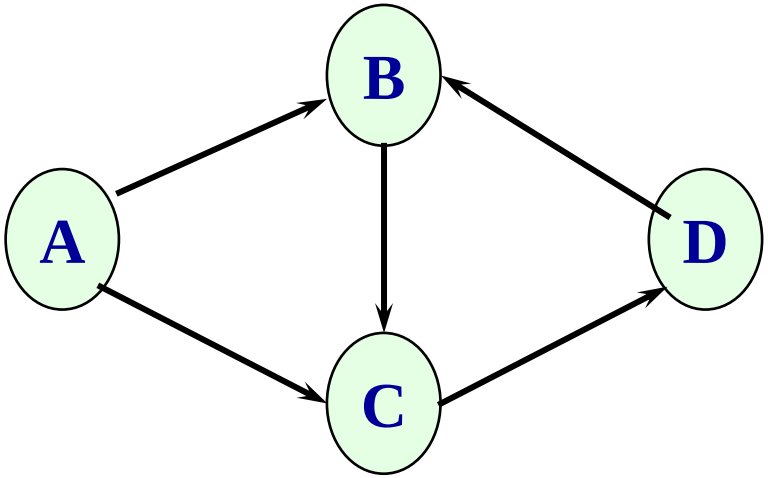
- 如果AOV网络中存在有向环，此AOV网络所代表的工程是不可行的。

练习：写出下图的拓扑排序序列。



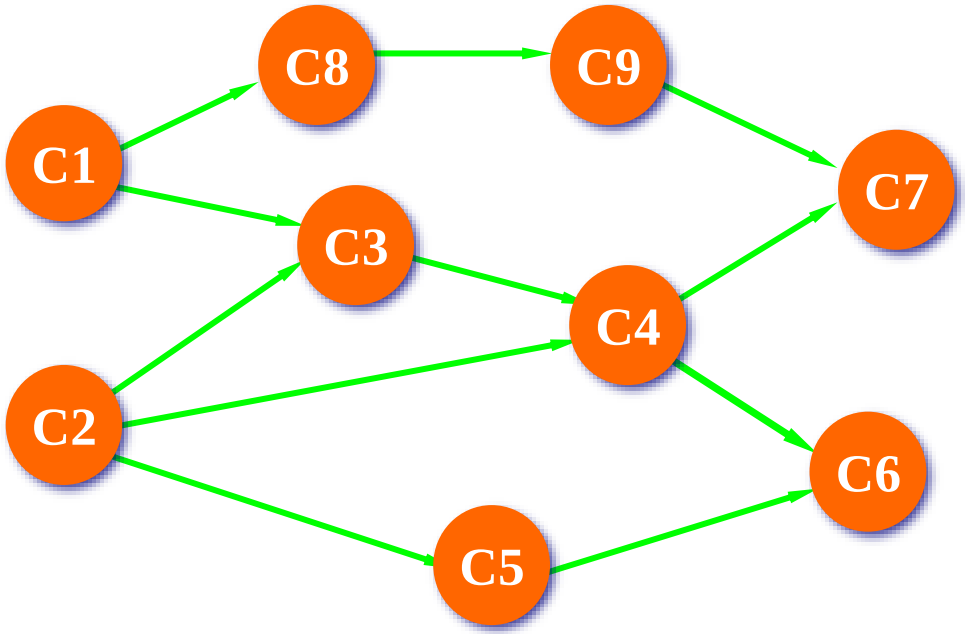
A B C D　或　A C B D

练习：写出下图的拓扑排序序列。



无拓扑排序序列，因为图中存在环 {B, C, D}。

例如，对学生选课工程图进行拓扑排序，　得到的拓扑有序序列为：



$C_1, C_2, C_3, C_4, C_5, C_6, C_8, C_9, C_7$

或 $C_1, C_8, C_9, C_2, C_5, C_3, C_4, C_7, C_6$

拓扑排序实现

实现所涉及的三个问题

没有前驱的顶点 == 入度为零的顶点

删除顶点及以它为尾的弧 == 弧头顶点的入度减1

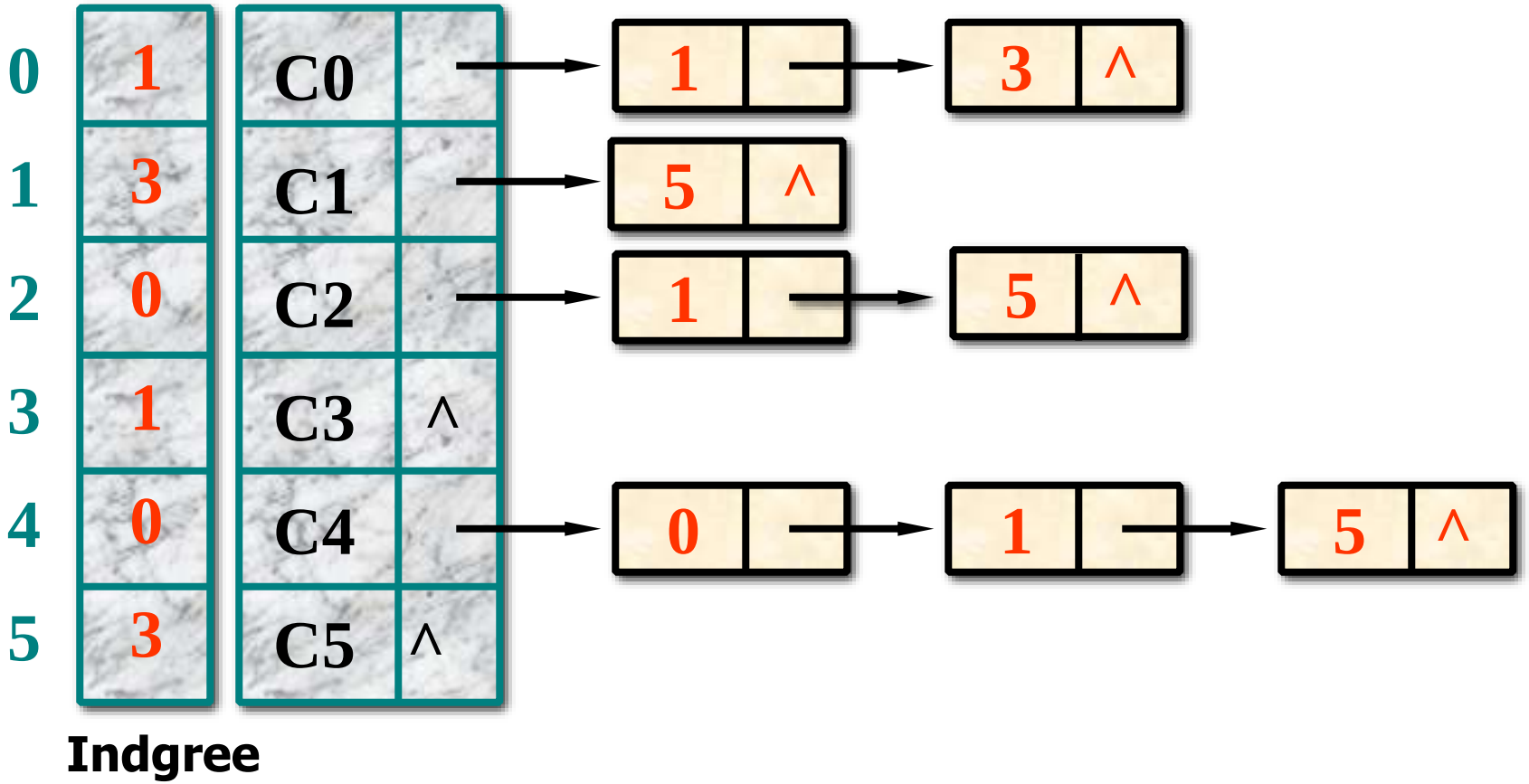
如何选择入度为零的顶点呢？ **== 栈或队列**

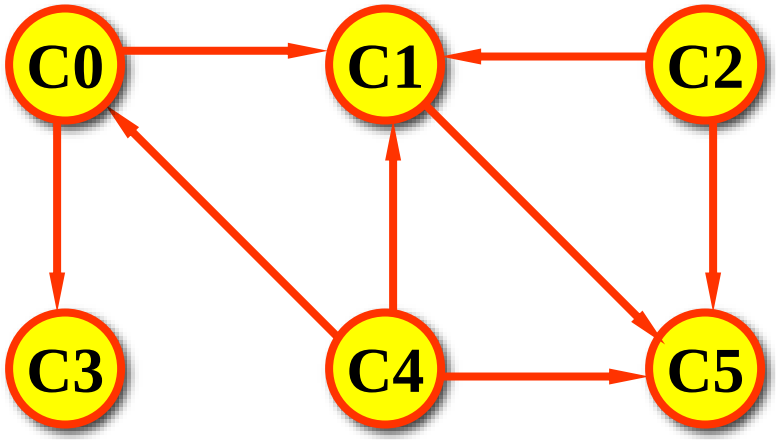
数组：**InDegree[ ],** 记录各顶点入度

```
class ALGraph {
  private:
      VexNode      *vertices;
      int          ArcNum;
      int          VexNum;
      int          GKind;
      int          GetLocVex(char  vex[]);
      void         BFS(char vex[],bool visited[]);
      void         DFS(char vex[],bool visited[]);
  public:
      ALGraph(){};
      void CreateALGraph();
      void BFSTraverse( );
      void DFTraverse( );
      void Prim(char vex[]);
   void Kruskal();
   void Dijkstra(char var[]);
   //从顶点var出发到各顶点的最短路径及路径值,
   bool TopologicalSort();};
```

```
bool ALGraph::TopologicalSort()
{
    stack<int>          q;
    Indgree = FindeInDegree();//计算各顶点入度
    for(i=0; i<vexnum; i++)    //入度为0的顶点入栈
        if(!Indgree[i])   q.push(i);
    while(!q.empty) {              //栈不空
        i = q.top(); q.pop();  count++;
        cout<< vertices[i].vertex;    //输出栈顶
        for(p=vertices[i].firstarc; p; p=p->next) {
            Indgree[p->adjvex]--;      //修改入度
            if(!Indgree[p->adjvex])
                q.push(p->adjvex);
        }
    if(count<vexnum)   return false;
    return ture;
}
```

练习：写出某AOV网的邻接表存储结构如下，写出拓扑排序序列。

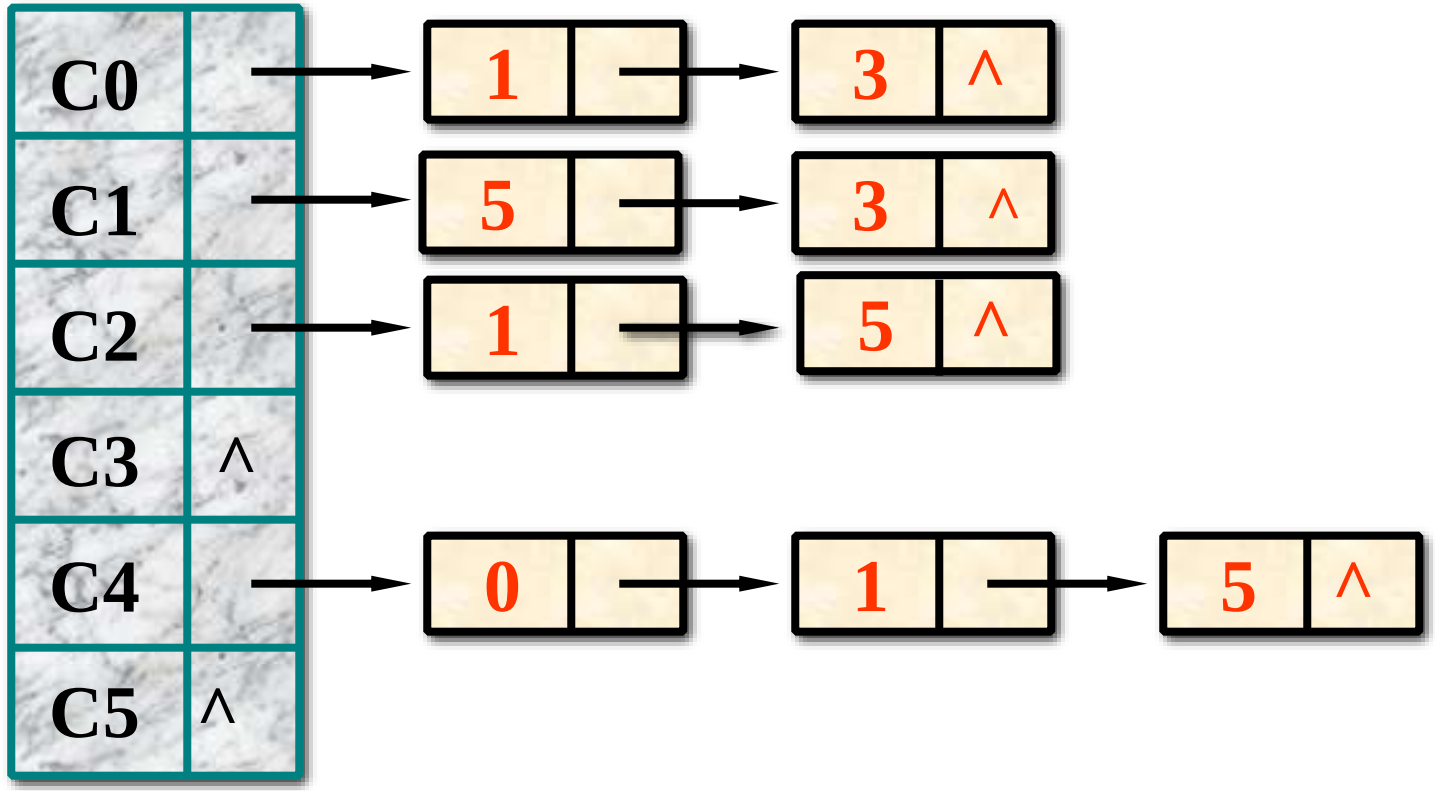




拓扑排序：C4、C0、C3、C2、C1、C5（栈）

C2、C4、C0、C1、C3、C5（队列）

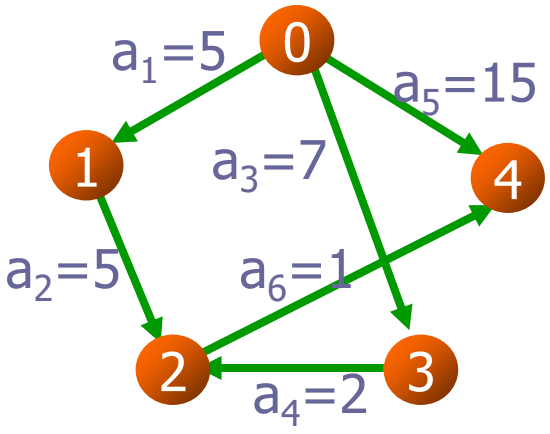
作业：写出某AOV网的邻接表存储结构如下，写出**分别用队列和栈**存储入读为零的顶点时的拓扑排序序列。



第六节　有向无环图及其应用

四、AOE-网

- 如果在无有向环的带权有向图中，用有向边表示一个工程中的活动（Activity），用边上权值表示活动持续时间（Duration），用顶点表示事件（Event），则这样的有向图叫做用边表示活动的网络，简称 AOE（Activity On Edges）网络。
- AOE应该同样是DAG
- AOE包括估算工程的完成时间

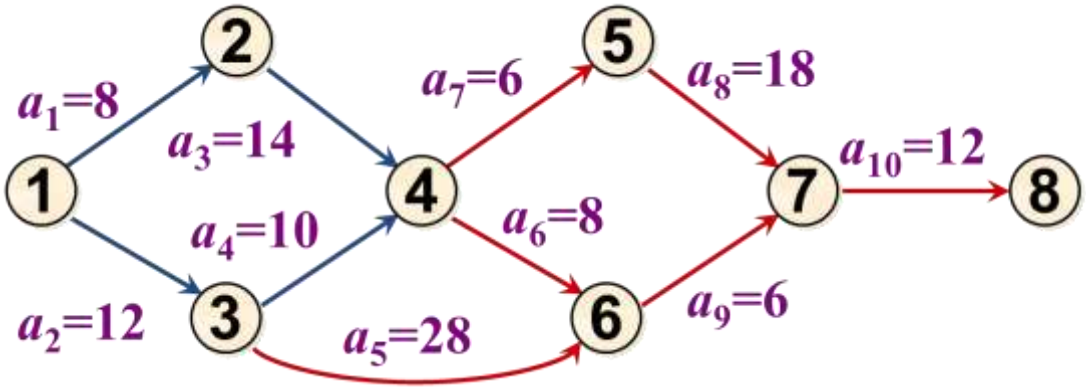


第六节　有向无环图及其应用

五、关键路径

1. 关键路径

- 求工程的完成时间是AOE的一个应用
- 在工程问题中，需要研究的问题有：
  完成整个工程至少需要多少时间？
  哪些活动是影响工程进度的关键？

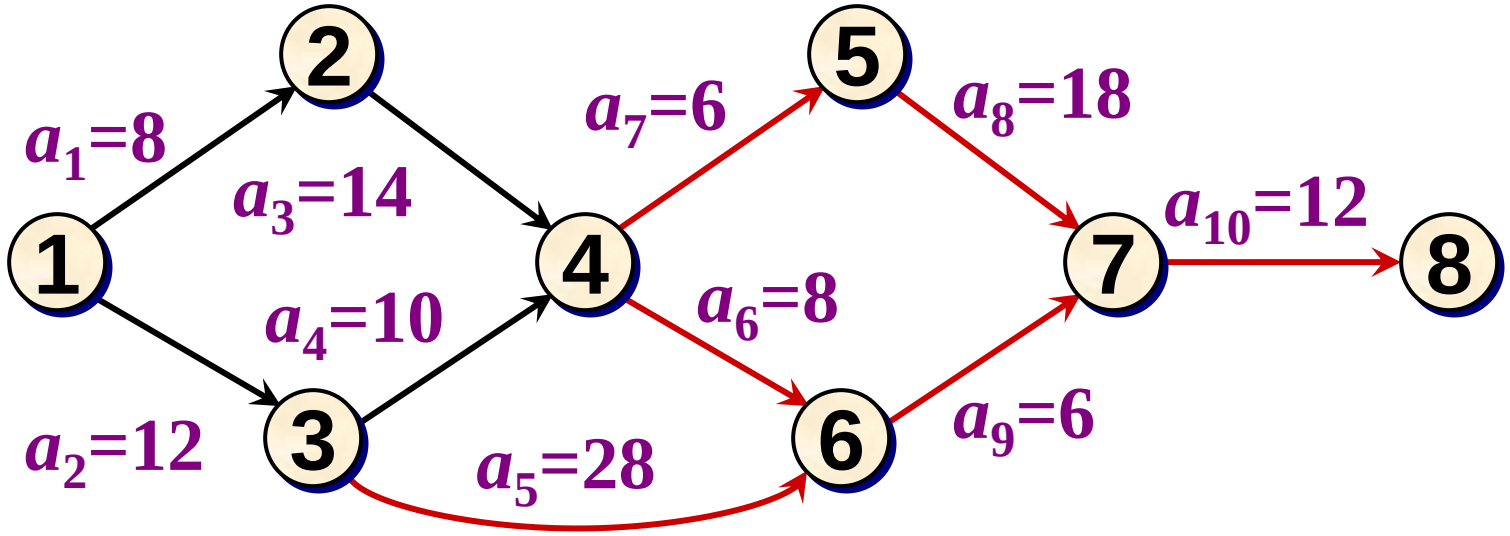


- 整个工程只有一个开始点和一个完成点，开始点（即入度为零的顶点）称为源点，结束点（即出度为零的顶点）称为汇点。

- 从源点到各个顶点、以及从源点到汇点的有向路径可能不止一条。这些路径的长度也可能不同。完成不同路径的活动所需的时间虽然不同，但只有各条路径上所有活动都完成了，整个工程才算完成。

- 因此，完成整个工程所需的时间取决于从源点到汇点的最长路径长度，即在这条路径上所有活动的持续时间之和。这条路径长度最长的路径就叫做关键路径(Critical Path)。

- 要找出关键路径，必须找出**关键活动**，即不按期完成就会影响整个工程完成的活动。
- 关键路径上的所有活动都是关键活动。因此，只要找到了关键活动，就可以找到关键路径。
- 例如，下图是一个AOE网。

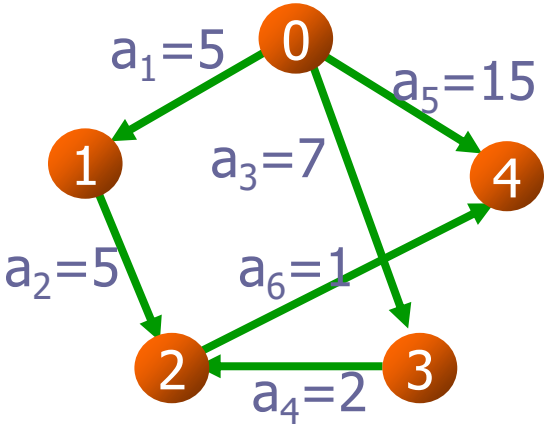


第六节　有向无环图及其应用

五、关键路径

2. 关键活动

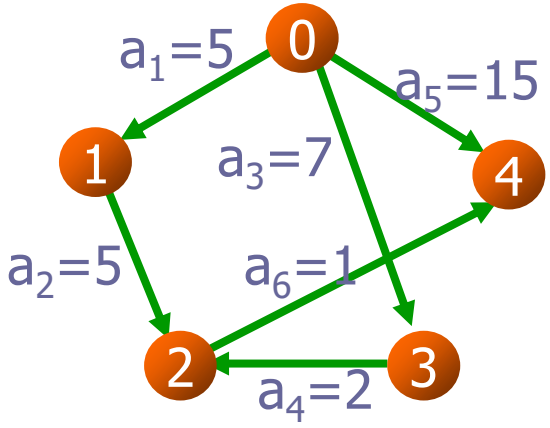
- e(i)：活动a_i最早开始时间
- l(i)：活动a_i最迟开始时间



- l(i)-e(i)：活动a_i开始时间余量
- 如果l(i)=e(i)，则称活动a_i为**关键活动**

第六节　有向无环图及其应用

五、关键路径



3. 关键活动有关的量

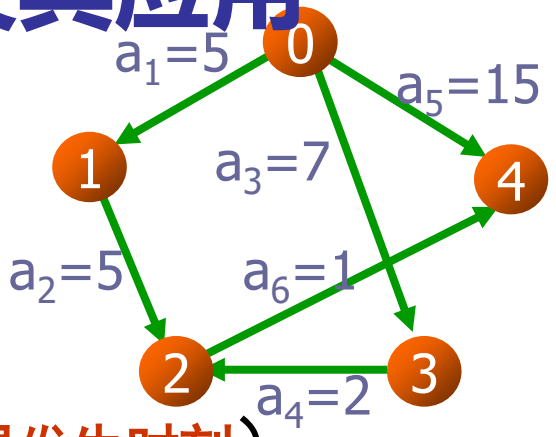
- ve(j)：事件v_j最早开始时间
- vl(j)：事件v_j最迟开始时间
- e(i)=ve(j)
- l(i)=vl(k)-dut(<j, k>)

  dut(<j, k>)为活动a_i的持续时间

•活动的最早开始时间是活动的弧尾事件的最早发生时间
•活动的最晚发生时间是活动的弧头事件的最晚发生时间减去活动的持续时间

第六节　有向无环图及其应用

五、关键路径

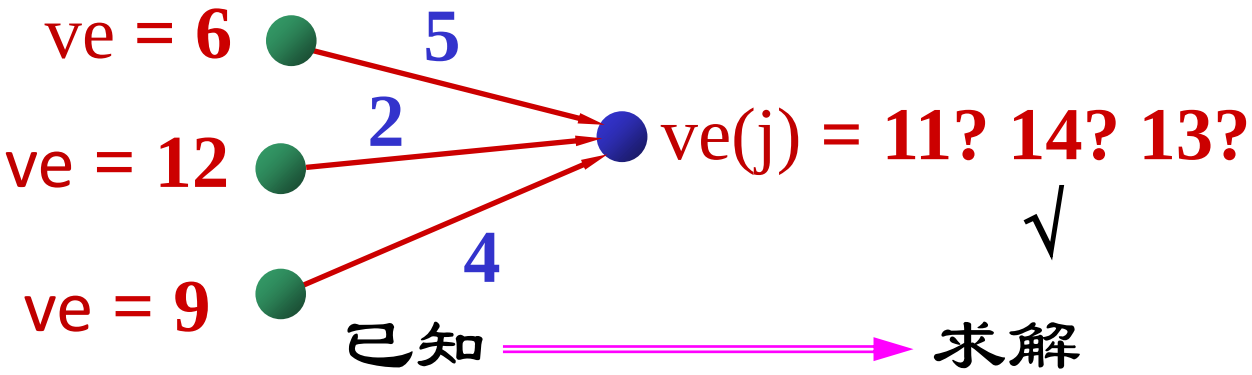


3. 关键活动有关的量

- 从ve(0)=0开始向前递推（事件的最早发生时刻）

  ve(j)=Max{ve(i)+dut(<i, j>)}

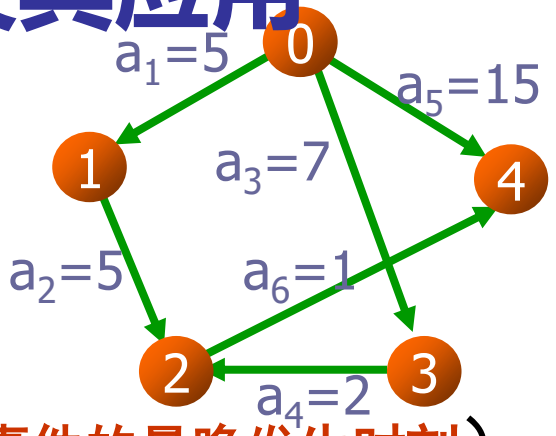
  <i, j>∈T，T是所有以第j个顶点为头的弧的集合



事件的最早发生时间是以其为弧头事件的所有弧尾事件的最早发生时间与对应弧活动的持续时间之和的最大值

第六节　有向无环图及其应用

五、关键路径

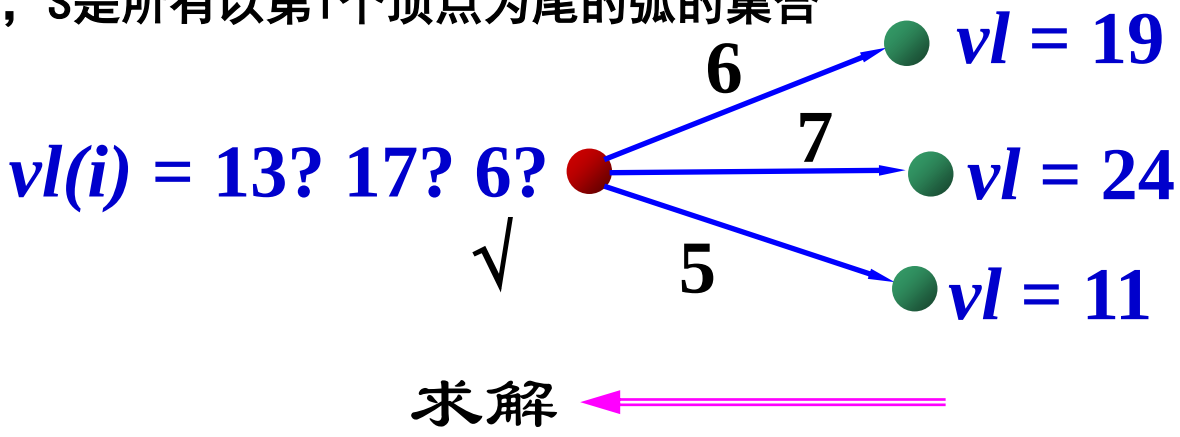


3. 关键活动有关的量

- 从vl(n-1)=ve(n-1)起向后递推（事件的最晚发生时刻）

  vl(i)=Min{vl(j)-dut(<i, j>)}

<i, j>∈S，S是所有以第i个顶点为尾的弧的集合



求解

事件的最晚发生时间是以其为弧尾事件的所有弧头事件的最晚发生时间与对应弧活动的持续时间之差的最小值

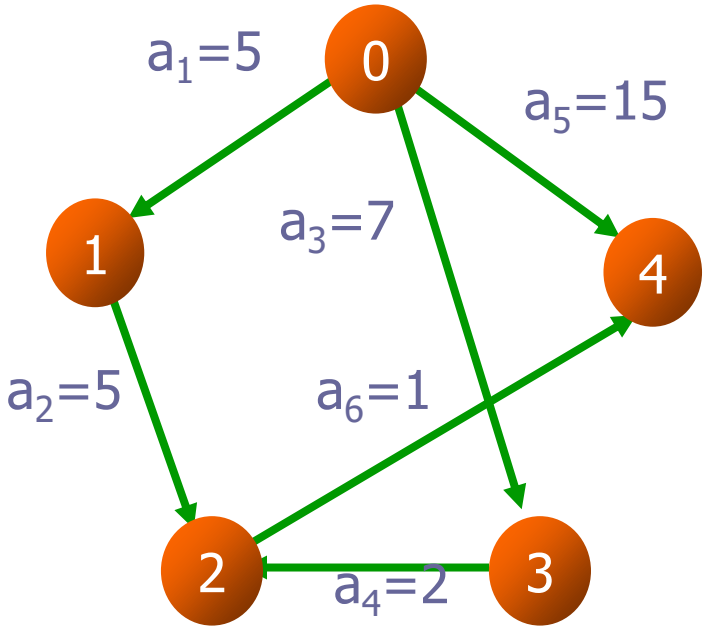
第六节　有向无环图及其应用

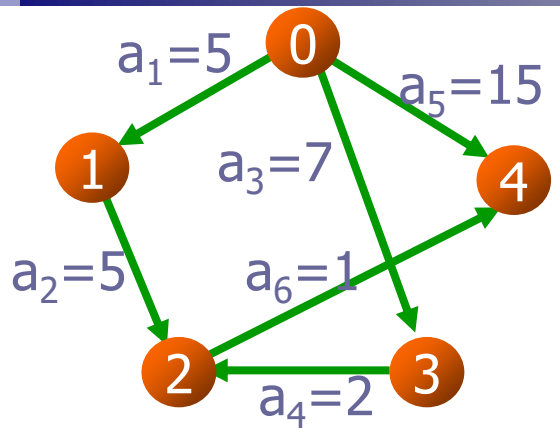
五、关键路径

4. 求关键活动算法（先计算事件，再计算活动）

- 从始点v_0出发，令ve[0]=0，按拓扑有序求ve[j]
- 从终点v_{n-1}出发，令vl[n-1]=ve[n-1]，按逆拓扑有序求vl[i]
- 根据各顶点的ve和vl值，求每条弧(活动)a_i的最早开始时间e[a_i]和最迟开始时间l[a_i]
- 如果e[a_i]=l[a_i]，则a_i为关键活动
- 如果ve[i]=vl[i]，则vi为关键路径上的事件

练习：求下图的关键路径。





拓扑有序　0,1,3,2,4

ve(j)=Max{ve(i)+dut(<i, j>)}

vl(i)=Min{vl(j)-dut(<i, j>)}

顶点	ve	vl
0	0	0
1	5	9
2	10	14
3	7	12
4	15	15

e(i)=ve(j)

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

活动	e	l	l-e
a_1	0	4	4
a_2	5	9	4
a_3	0	5	5
a_4	7	12	5
a_5	0	0	0
a_6	10	14	4

练习：下表给出了某工程各工序之间的优先关系和各工序所需的时间。

工序代号	A	B	C	D	E	F	G	H
所需时间	3	2	2	3	4	3	2	1
先驱工序	-	-	A	A	B	B	C,E	D

问：该工程是否能够顺利进行？

如果能，请问要花多长时间？

缩短那些工序可以缩短整个工程的完工时间？