数据结构与算法设计

2021-09

课程内容简介

第1章 绪论	第8章 排序与分治	串与串匹配算法
第2章 线性表	第9章 外部排序	红黑树
第3章 栈和队列	第10章 动态规划算法	k-d树
第4章 数组和广义表	第11章 有限自动机	复杂图算法
第5章 树、二叉树、回溯法	第12章 图灵机	文本检索技术
第6章 图与贪心算法	第13章可判定性	分支限界算法
第7章 查找	第14章 时间复杂性	随机化算法
		上下文无关文法

6.1 图的术语与定义

图的定义

图(Graph)——图G是由两个集合V(G)和E(G)组成的,记为G=(V,E)

其中：V(G)是顶点的非空有限集

E(G)是边的有限集合，边是顶点的无序对或有序对。

图的分类

有向图

无向图

6.1 图的术语与定义

图的定义

有向图——有向图G是由两个集合V(G)和E(G)组成的。

其中：

V(G)是顶点的非空有限集。

E(G)是有向边（也称弧）的有限集合，弧是顶点的有序对，记为<v,w>，v,w是顶点，v为弧尾，w为弧头。

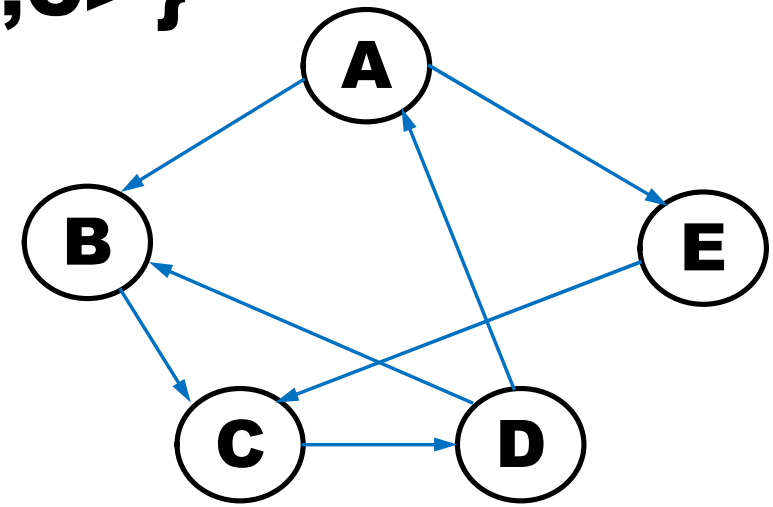
6.1 图的术语与定义

例如：

G1 = <V1, E1>

V1 = { A, B, C, D, E }

E1 = {<A,B>, <A,E>, <B,C>, <C,D>, <D,B>,
<D,A>, <E,C> }



6.1 图的术语与定义

图的定义

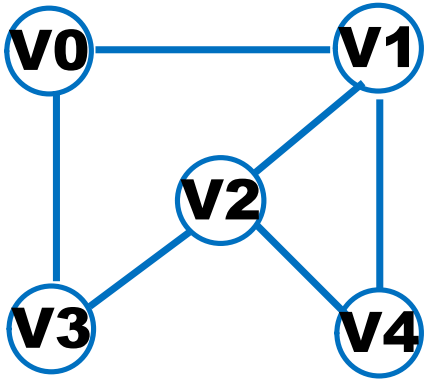
无向图——无向图G是由两个集合V(G)和E(G)组成的。

其中：

V(G)是顶点的非空有限集。

E(G)是边的有限集合，边是顶点的无序对，记为 (v,w) 或 (w,v)，并且(v,w) = (w,v)。

6.1 图的术语与定义



例如：

G2 = <V2, E2>

V2 = { v0, v1, v2, v3, v4 }

E2 = { (v0,v1), (v0,v3), (v1,v2), (v1,v4),
(v2,v3), (v2,v4) }

6.1 图的术语与定义

图的应用举例

例1. 交通图（公路、铁路）

顶点：地点　　边：连接地点的路

例2. 电路图

顶点：元件　　边：连接元件之间的线路

例3. 通讯线路图

顶点：地点　　边：地点间的连线

例4. 各种流程图

如产品的生产流程图。

顶点：工序　　边：各道工序之间的顺序关系

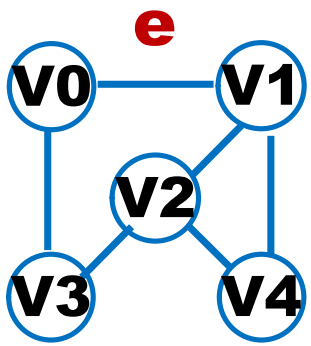
6.1 图的术语与定义

图的基本术语

邻接点及关联边

邻接点：边的两个顶点

关联边：若边 e = (v, u), 则称顶点v、u 关连边 e。



6.1 图的术语与定义

顶点的度、入度、出度

顶点v的度 = 与v相关联的边的数目

在有向图中：

顶点v的出度 = 以v为起点有向边数

顶点v的入度 = 以v为终点有向边数

顶点v的度 = v的出度+v的入度

设图G 的顶点数为 n，边数为 e

图的所有顶点的度数和 = 2*e

（每条边对图的所有顶点的度数和“贡献”2度）

6.1 图的术语与定义

路径、回路

无向图 G =（V，E）中的顶点序列$v_1, v_2, \cdots, v_k$，若$(v_i, v_{i+1}) \in E$（i=1, 2 ,$\cdots$ k-1），v=v_1，u=v_k，则称该序列是从顶点v到顶点u的路径；若v=u，则称该序列为回路。

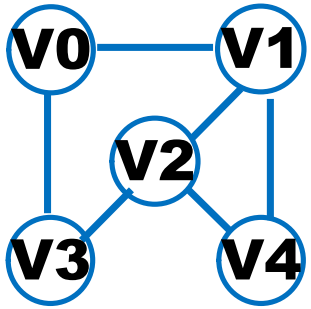
有向图 D =（V，E）中的顶点序列 $v_1, v_2, \cdots, v_k$，若$\langle v_i, v_{i+1} \rangle \in E$（i=1,2,$\cdots$k-1），v=$v_1$，u=$v_k$，则称该序列是从顶点v到顶点u的路径；若v=u，则称该序列为回路。

6.1 图的术语与定义

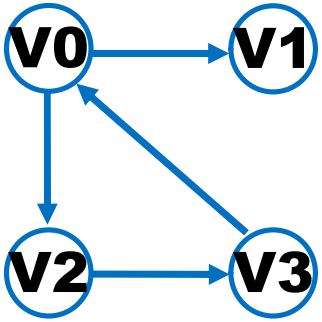
例如

在图G1中，V0, V1, V2, V3 是 V0 到 V3 的路径；V0, V1, V2, V3, V0 是回路。

在图G2中，V0, V2, V3 是 V0 到 V3 的路径；V0, V2, V3, V0 是回路。



无向图G1



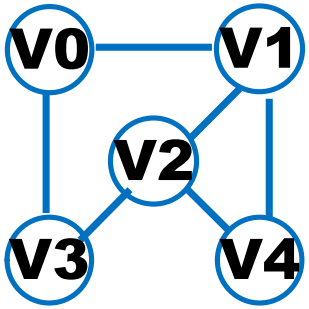
有向图G2

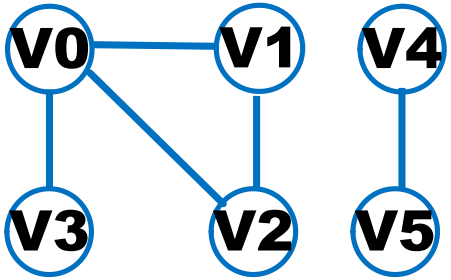
6.1 图的术语与定义

连通图（强连通图）

在无（有）向图 G=< V, E > 中，若对任何两个顶点 v、u 都存在从 v 到 u 的路径，则称G是连通图（强连通图）。

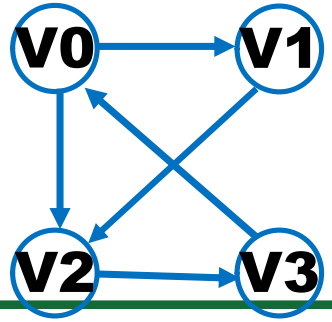
连通图

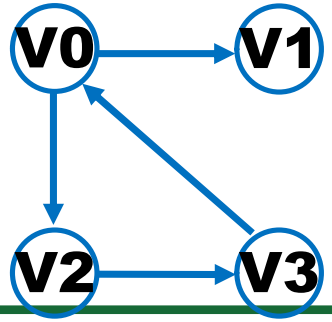




非连通图

强连通图





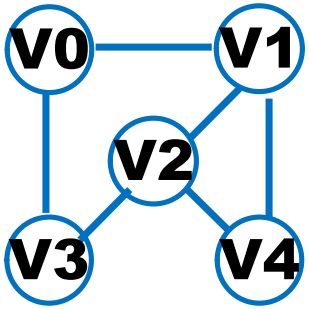
非强连通图

6.1 图的术语与定义

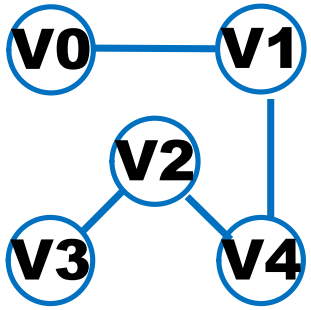
子图

设有两个图 G=(V，E)，G1=(V1，E1)，若 V1 ⊆ V，E1 ⊆ E，E1关联的顶点都在 V1 中，则称 G1 是 G 的子图。

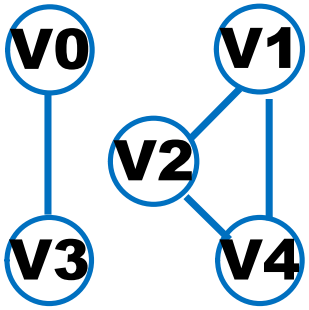
例　(b)、(c) 是 (a) 的子图



(a)



(b)

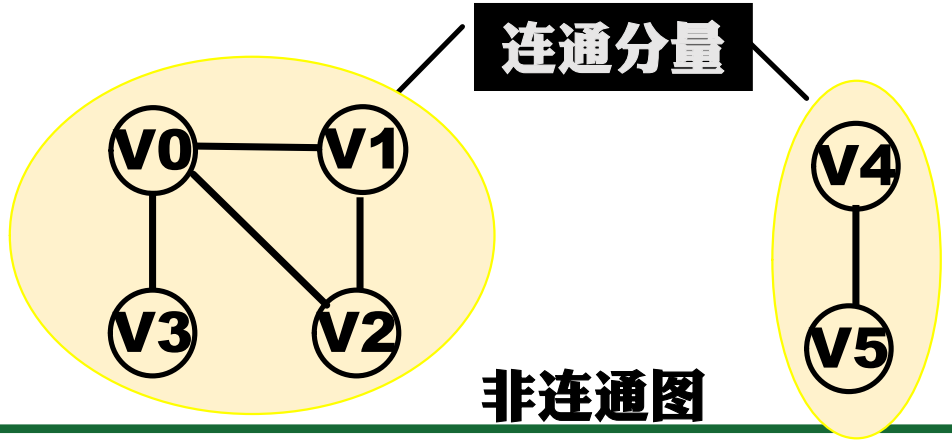


(c)

6.1 图的术语与定义

连通分量

无向图G的极大连通子图称为G的连通分量。任意两个顶点之间都有路径相通，则称此图为连通图(Connected graph)，极大连通子图含义：该子图是G连通子图，将G的任何不在该子图中的顶点加入，子图不再连通。



6.1 图的术语与定义

连通分量

连通图G的连通分量，只有一个，就是G本身。

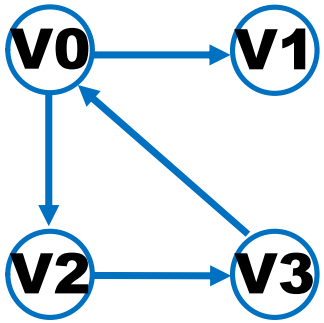
非连通图的连通分量，可以有多个。

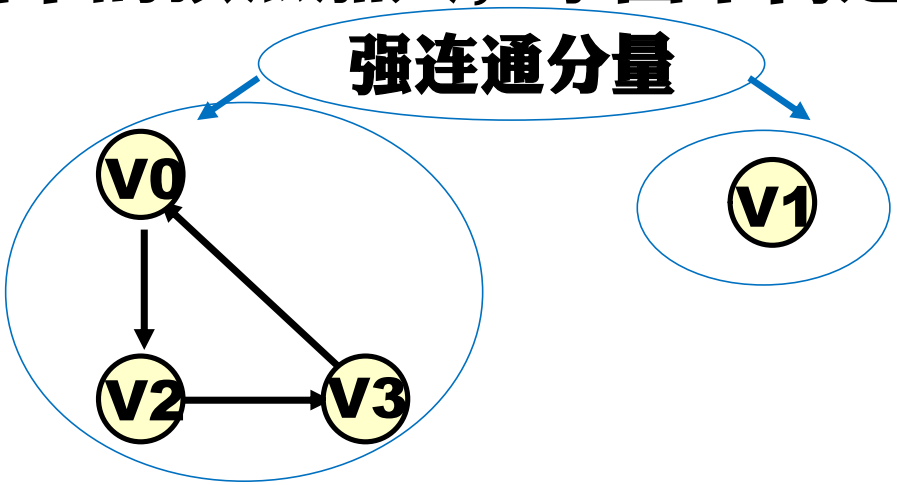
6.1 图的术语与定义

连通子图（强连通分量）若任意两个顶点之间都存在一条有向路径，则称此有向图为**强连通图**

有向图D的极大强连通子图称为D的强连通分量。

极大强连通子图含义：该子图是 D 的强连通子图，将 D 的任何不在该子图中的顶点加入，子图不再是强连通的。





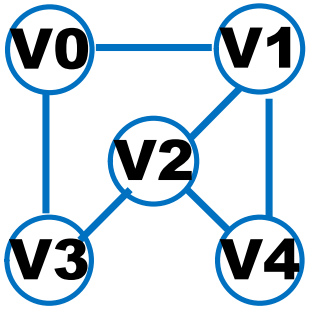
6.1 图的术语与定义

生成树

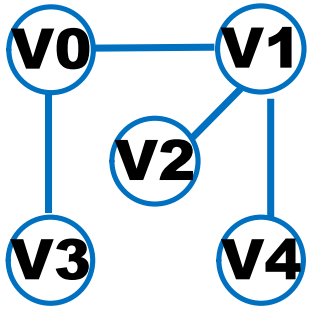
包含无向图 G 所有顶点的极小连通子图称为G生成树。

极小连通子图含义：该子图是G的连通子图，在该子图中删除任何一条边，子图不再连通，若T是G的生成树当且仅当T满足如下条件：

- T是G的连通子图
- T包含G的所有顶点
- T中无回路



连通图G1



G1的生成树

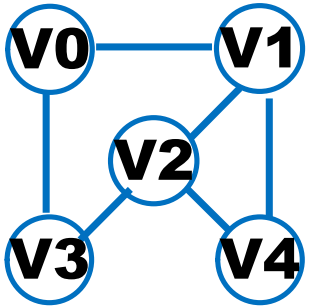
6.2 图的存储结构

一、数组表示法（邻接矩阵表示）

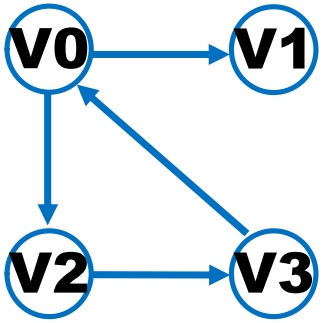
邻接矩阵： G的邻接矩阵是满足如下条件的n阶矩阵：

$$A[i][j]=\begin{cases}1 & \text{若}(v_i,v_j)\subseteq E \text{ 或 } \langle v_i,v_j\rangle\subseteq E\\ 0 & \text{否则}\end{cases}$$

在数组表示法中，用邻接矩阵表示顶点间的关系



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



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

6.2 图的存储结构

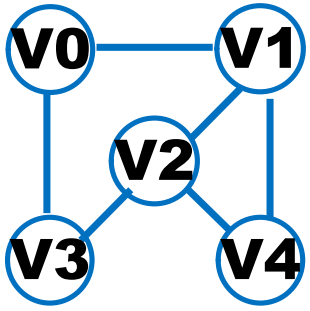
二、邻接表

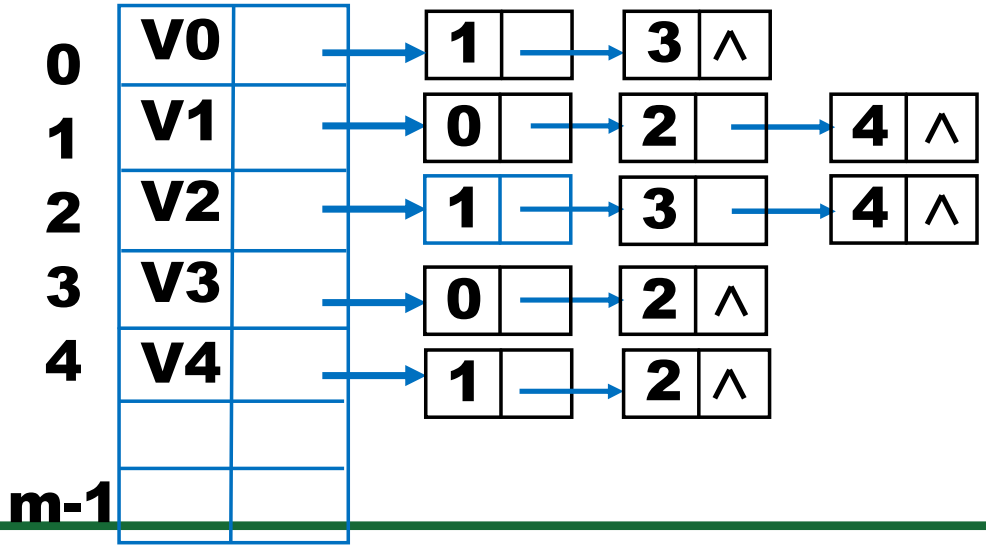
邻接表是图的链式存储结构

1、无向图的邻接表

顶点：通常按编号顺序将顶点数据存储在一维数组中；

关联同一顶点的边：用线性链表存储。





6.2 图的存储结构

```
typedef struct ArcNode  // 结点定义
    {  int  adjvex;                    // 邻接点域，
                                       // 存放与Vi邻接的点在表头数组中的位置
       struct  ArcNode  *next;  // 链域，下一条边或弧
    } ArcNode;
```

adjvex	next

```
    typedef struct tnode  // 表头结点
        {   int  vexdata;                  // 存放顶点信息
            ArcNode  * firstarc;           // 指向第一个邻接点
        } VNode, AdjList [ MAX_VERTEX_NUM ] ;
```

vexdata	firstarc

```
    typedef struct
        {     AdjList  vertices;
              int      vexnum, arcnum;   // 顶点数和弧数
              int      kind;                // 图的种类
        }
```

6.2 图的存储结构

无向图的邻接表的特点

1）在G邻接表中，同一条边对应两个结点；

2）顶点v的度：等于v对应线性链表的长度；

3）判定两顶点v ，u是否邻接：要看v对应线性链表中有无对应的结点。

4）在G中增减边：要在两个单链表插入、删除结点；

5）设存储顶点的一维数组大小为 m（m≥图的顶点数n），图的边数为 e，G占用存储空间为：m+2*e。G占用存储空间与G的顶点数、边数均有关；适用于边稀疏的图。

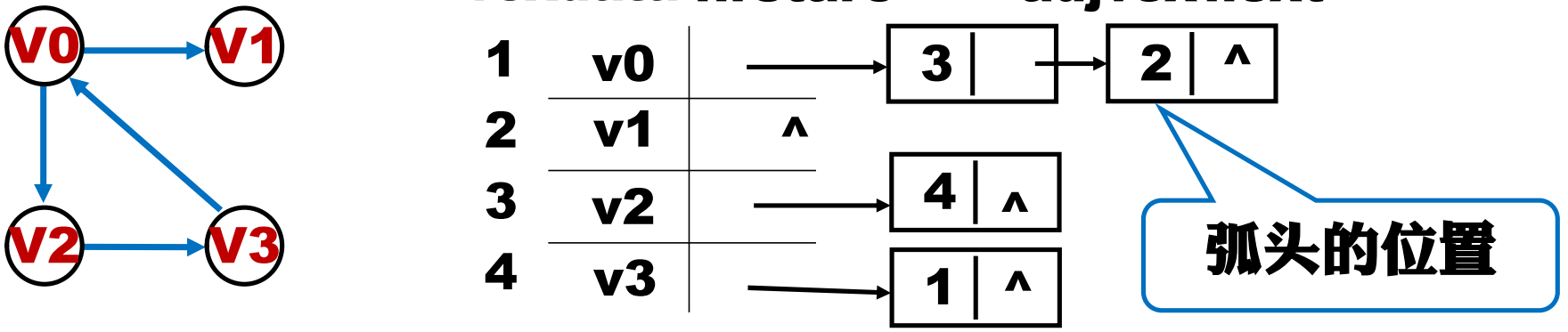
6.2 图的存储结构

二、邻接表

2、有向图的邻接表

顶点：用一维数组存储（按编号顺序）

以同一顶点为起点的弧：用线性链表存储



类似于无向图的邻接表，所不同的是：以同一顶点为起点的弧：用线性链表存储

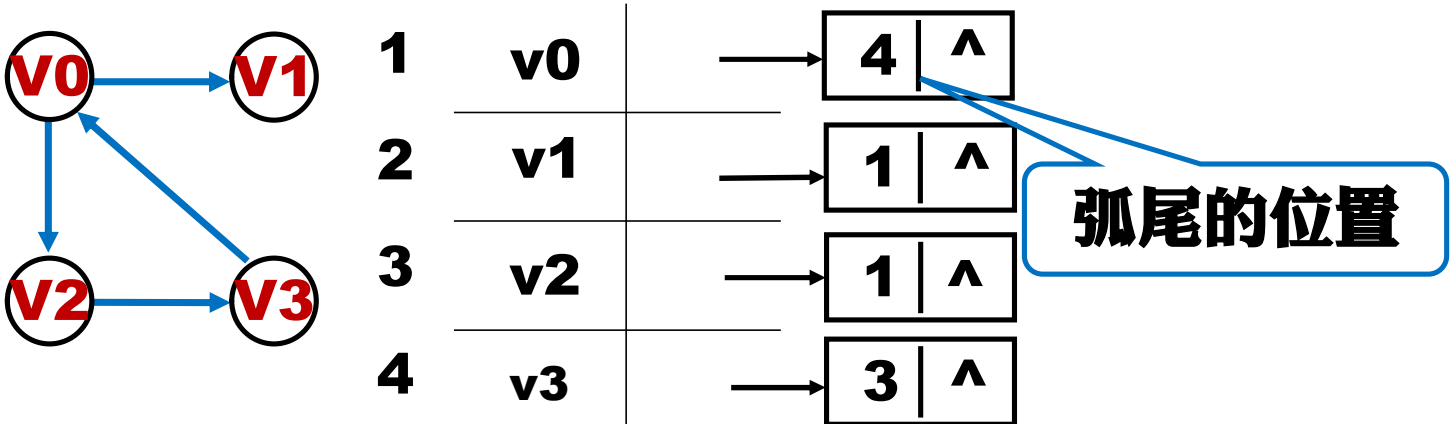
6.2 图的存储结构

二、邻接表

3、有向图的逆邻接表

顶点：用一维数组存储（按编号顺序）

以同一顶点为**终点**的弧：用线性链表存储。



类似于有向图的邻接表，所不同的是：

以同一顶点为**终点**弧：用线性链表存储

6.2 图的存储结构

三、有向图的十字链表表示法

弧结点：

```
typedef  struct  ArcBox
{   int   tailvex,  headvex;   // 弧尾、弧头在表头数组中位置
     struct   arcnode   *hlink;    // 指向弧头相同的下一条弧
     struct   arcnode   *tlink;     // 指向弧尾相同的下一条弧
} ArcBox;
```

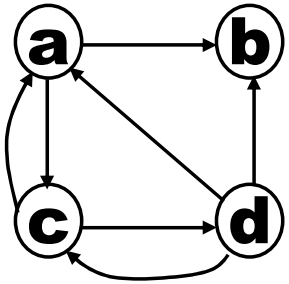
顶点结点：

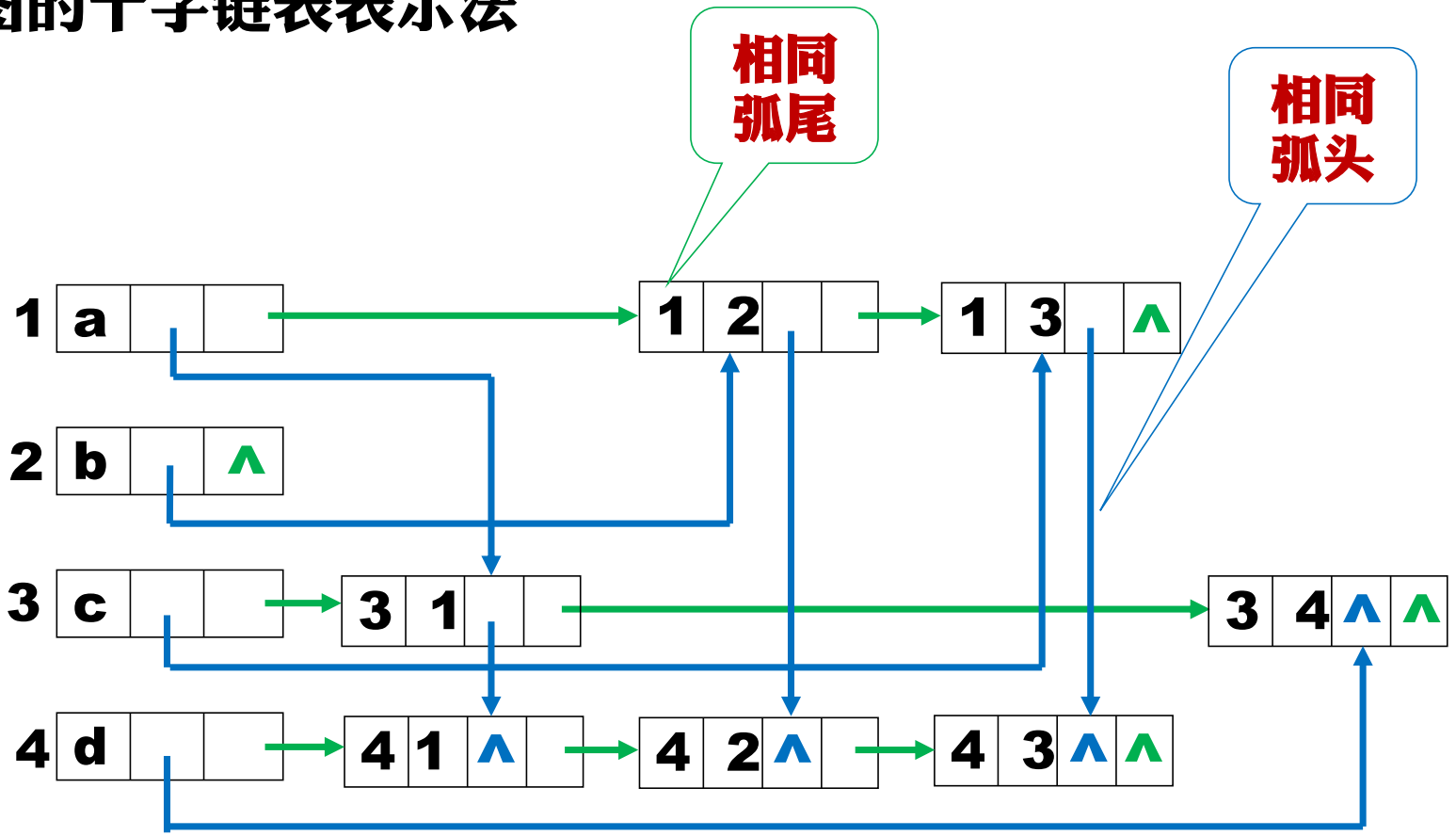
```
typedef  struct  VexNode
{  VertexType   data; // 存与顶点有关信息
    ArcBox *firstin;    // 指向以该顶点为弧头的第1个弧结点
    ArcBox *firstout;  // 指向以该顶点为弧尾的第1个弧结点
} VexNode;
VexNode  OLGraph[M];
```

6.2 图的存储结构

三、有向图的十字链表表示法

例





6.2 图的存储结构

四、无向图的邻接多重表表示法

边结点：

```
typedef   struct  node
{   VisitIf  mark;   // 标志域，记录是否已经搜索过
    int  ivex, jvex;  // 该边依附的两个顶点在表头数组中位置
    struct  EBox   * ilink,  * jlink;
                          //分别指向依附于ivex和jvex的下一条边
} EBox;
```

mark	ivex	ilink	jvex	jlink

顶点结点：

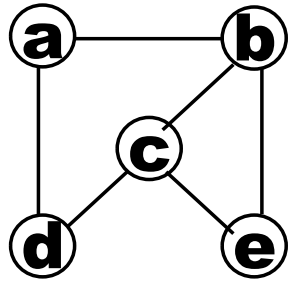
```
typedef   struct  VexBox
{   VertexType   data;          // 存与顶点有关的信息
    EBox  * firstedge;          // 指向第一条依附于该顶点的边
} VexBox;
VexBox  AMLGraph[M];
```

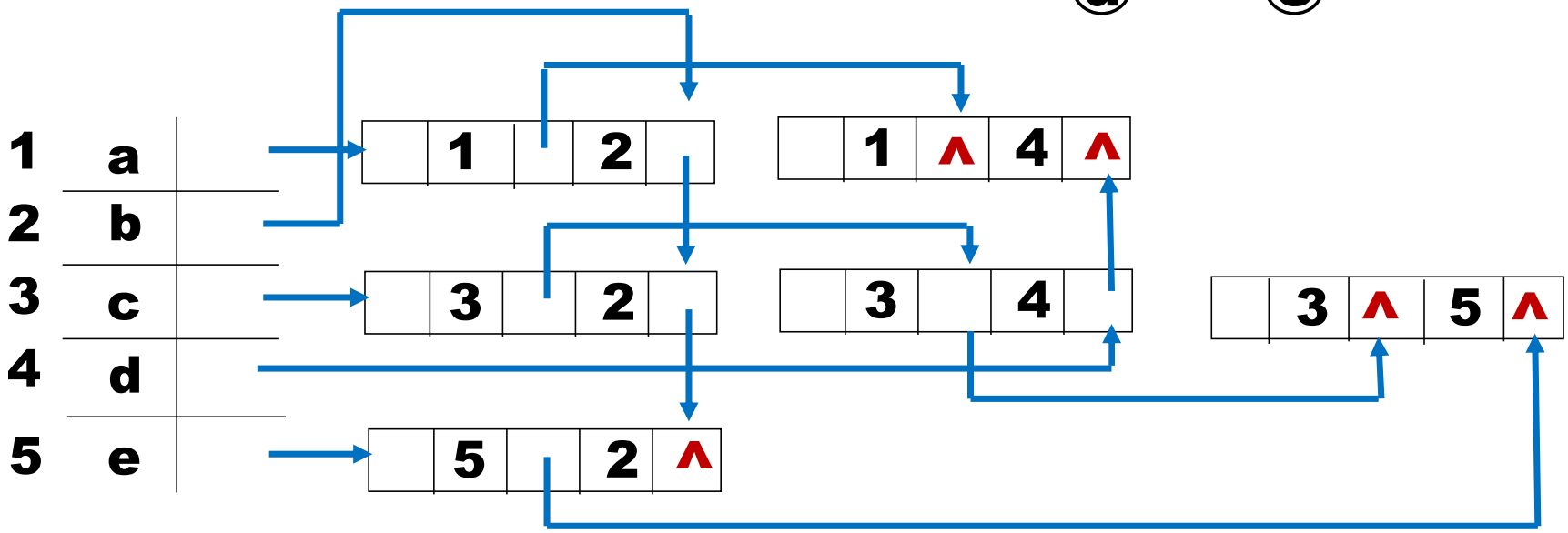
data	firstedge

6.2 图的存储结构

四、无向图的邻接多重表表示法

例





6.3 图的遍历

图的遍历

访遍图中所有的顶点，并且使图中的每个顶点仅被访问一次。

遍历实质

找每个顶点的邻接点。

搜索路径

深度优先遍历（DFS）

广度优先遍历（BFS）

6.3 图的遍历

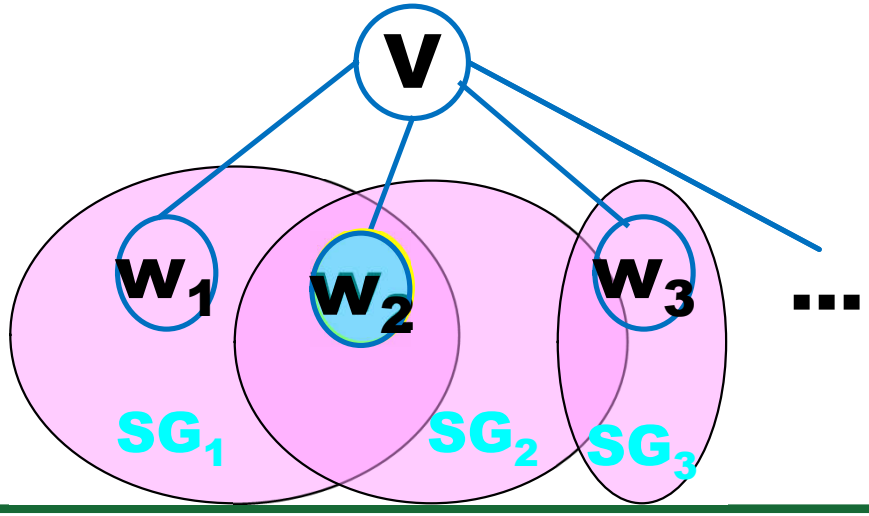
图的深度遍历（DFS）

从图的某顶点v出发，进行深度优先遍历

访问顶点 V ；

for (V的所有邻接点W_1、W_2、W_3...)

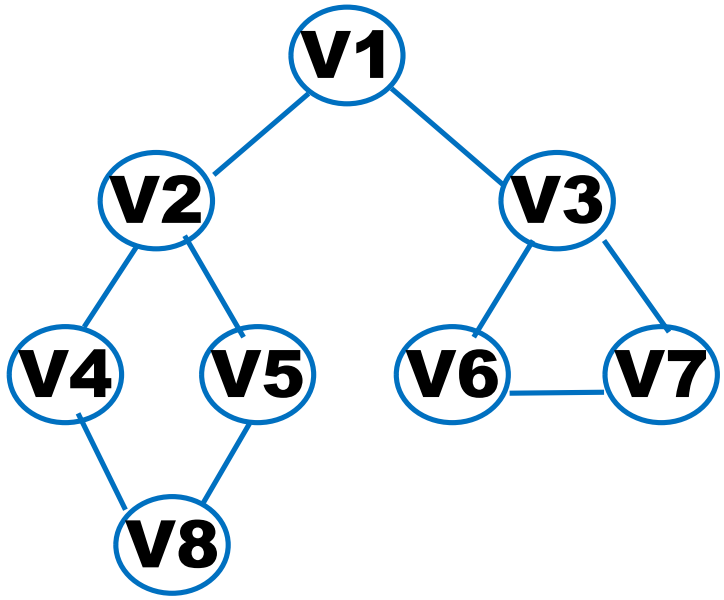
若 W_i 未被访问，则从 W_i 出发，进行深度优先遍历。



6.3 图的遍历

图的深度遍历（DFS）

例：

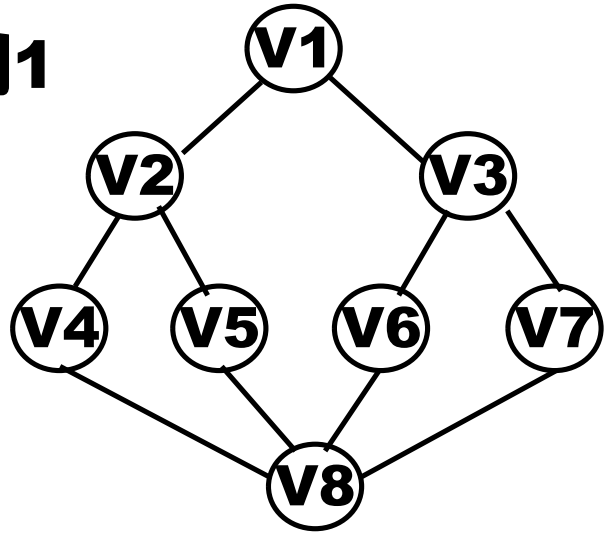


深度遍历： V1 ⇒ V2 ⇒V4 ⇒ V8 ⇒ V5 ⇒ V3 ⇒ V6 ⇒ V7

6.3 图的遍历

图的深度遍历（DFS）

例1



深度遍历1：V1 ⇒ V2 ⇒ V4 ⇒ V8 ⇒ V5 ⇒ V6 ⇒ V3 ⇒ V7

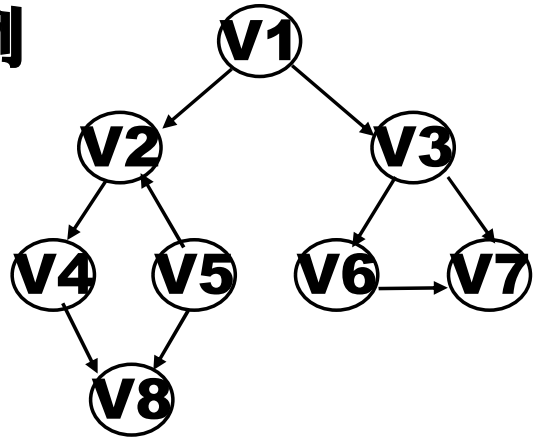
深度遍历2：V1 ⇒ V3 ⇒ V7 ⇒ V8 ⇒ V6 ⇒ V5 ⇒ V2 ⇒ V4

由于**没有规定访问邻接点的顺序**，所以深度优先序列不惟一。

6.3 图的遍历

图的深度遍历（DFS）

例

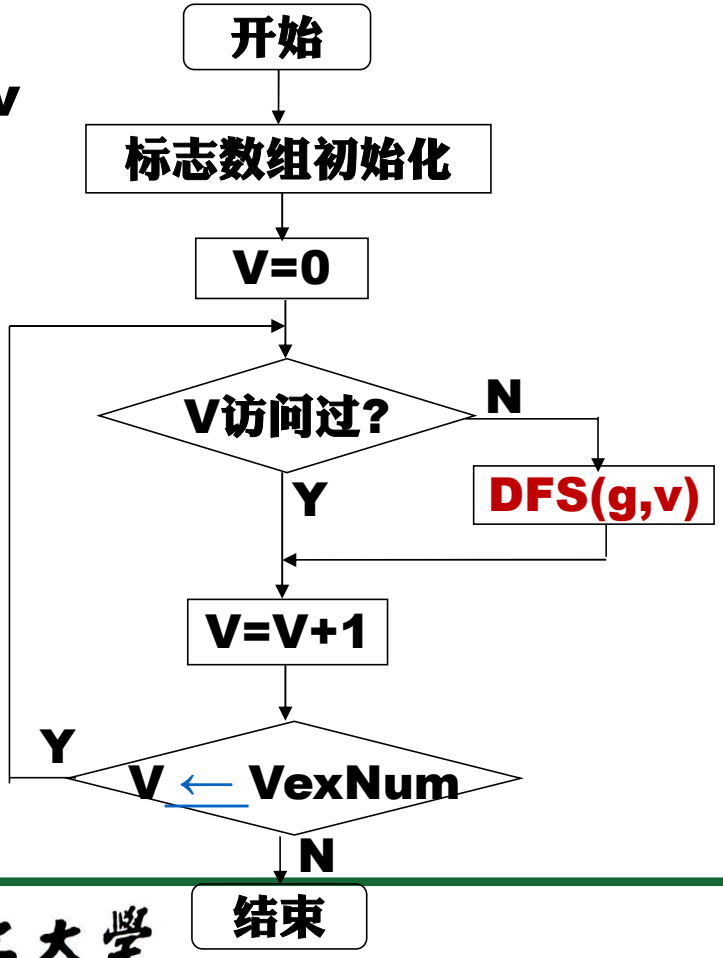


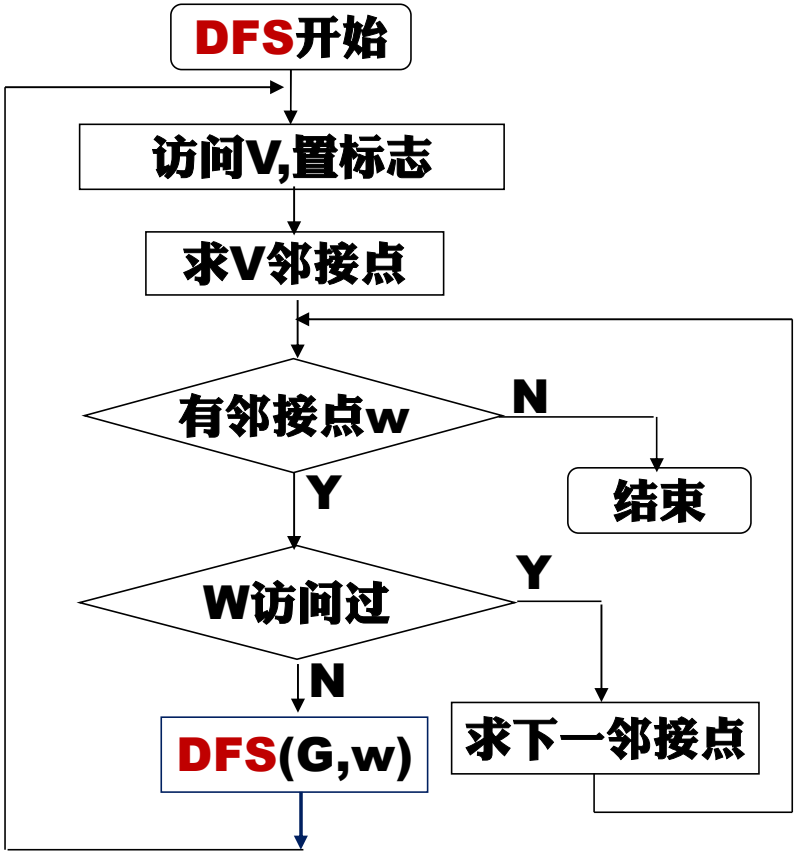
深度遍历：V1 ⇒ V2 ⇒ V4 ⇒ V8 ⇒ V3 ⇒ V6 ⇒ V7 ⇒ V5

6.3 图的遍历

图的深度遍历（DFS）——算法6.4和6.5

DFSTrav





6.3 图的遍历

图的深度遍历（DFS）——递归算法

```
void DFSTrav ( Graph G, Void ( * Visit ) ( VertexType e ) )
{
   for ( v=0; v< G.vexnum; ++v )
         visited[v] = FALSE;
   for ( v=0; v<G.vexnum; ++v )
         if ( ! visited[ v ] )
              DFS( G, v, Visit );
} //DFSTrav
```

6.3 图的遍历

图的深度遍历（DFS）——递归算法

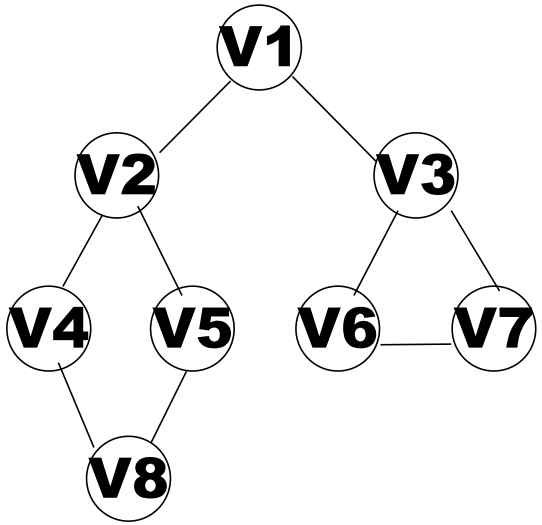
```
void DFS( Graph G, int v, void ( * Visit ) ( VertexType e ) )
{  /* 从v出发（v是顶点位置），深度优先遍历v所在的连通分量 */
   Visit( v );      //先根遍历
   visited[v] = TRUE;
   for ( w = FirstAdjVex( G, v ); w; w = NextAdjVex( G, v, w ) )
         if ( ! visited[ w ] )
             DFS( G, w, Visit( w ) );
} //DFS
```

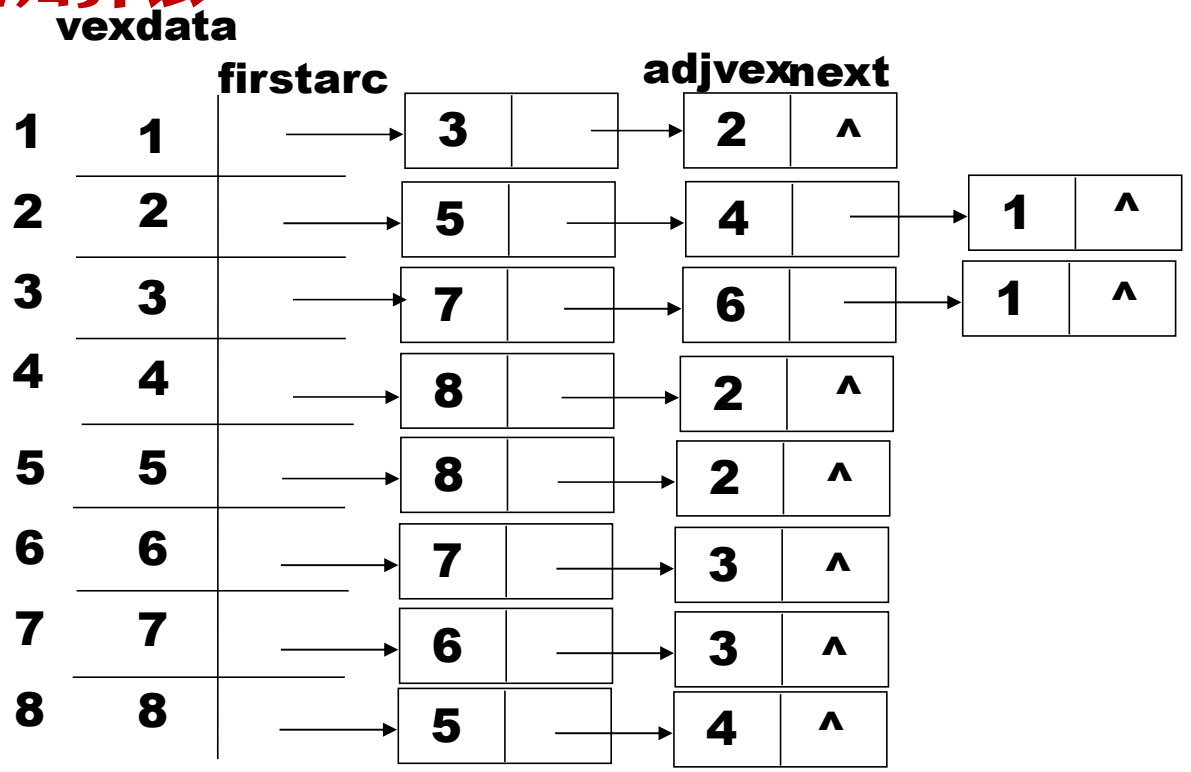
访问标志数组：int visited[ ] 全局变量，初始时所有分量全为FALSE

6.3 图的遍历

图的深度遍历（DFS）——递归算法

例



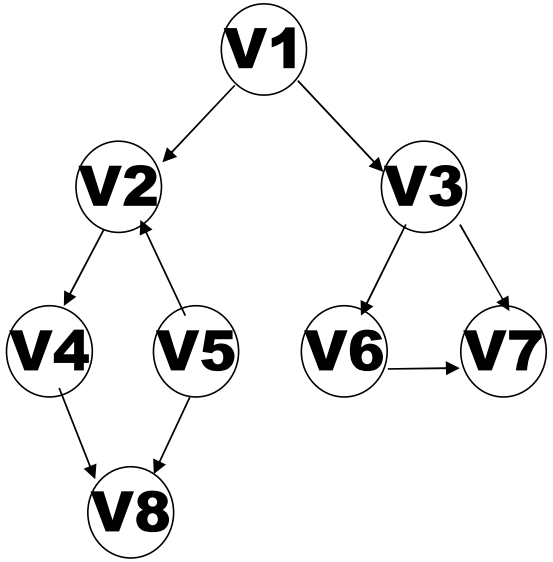


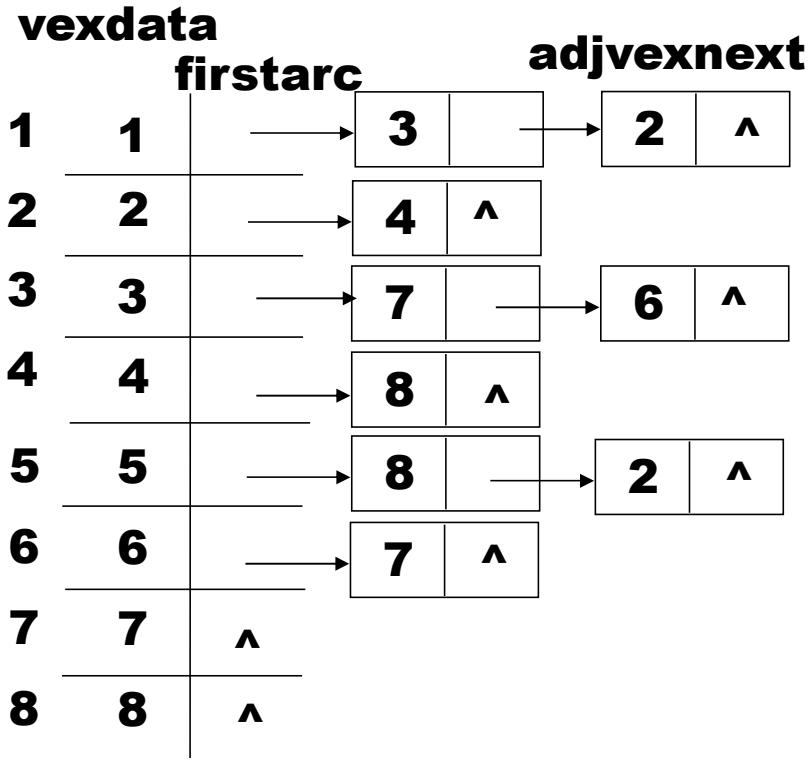
深度遍历：V1 ⇒ V3 ⇒ V7 ⇒ V6 ⇒ V2 ⇒ V5 ⇒ V8 ⇒ V4

6.3 图的遍历

图的深度遍历（DFS）——递归算法

例





深度遍历：V1 ⇒ V3 ⇒ V7 ⇒ V6 ⇒ V2 ⇒ V4 ⇒ V8 ⇒ V5

6.3 图的遍历

深度优先遍历的时间复杂度

DFS对每一条边处理一次，每个顶点访问一次。

邻接表表示总代价为：O(点数n + 边数e)

邻接矩阵表示：处理所有的边需要 $O(n^2)$ 的时间，所以总代价为 $O(n+n^2)= O(n^2)$。

6.3 图的遍历

图的广度遍历（BFS）

从图中某顶点v出发：

1)访问顶点v；

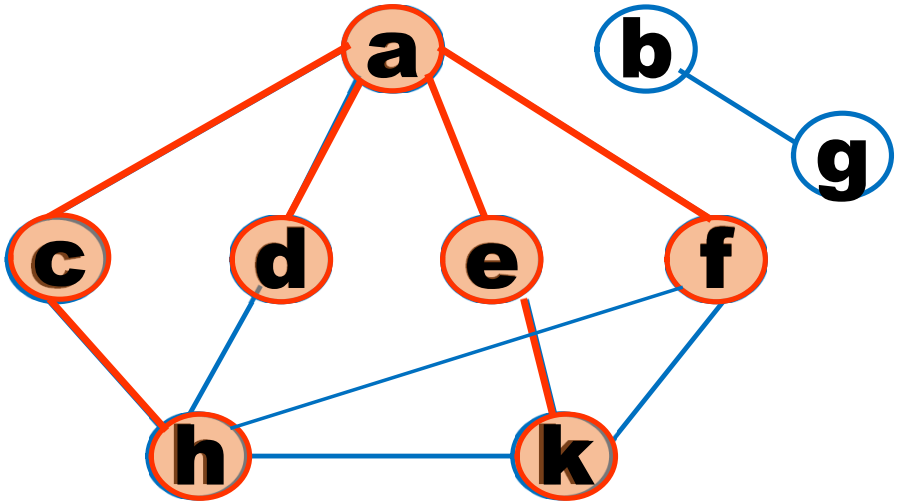
2)访问v所有未被访问的邻接点w1,w2,...wk；

3)依次从这些邻接点出发，访问其所有未被访问的邻接点。依此类推，直至图中所有和V0有路径相通的顶点都被访问到。

6.3 图的遍历

图的广度遍历（BFS）

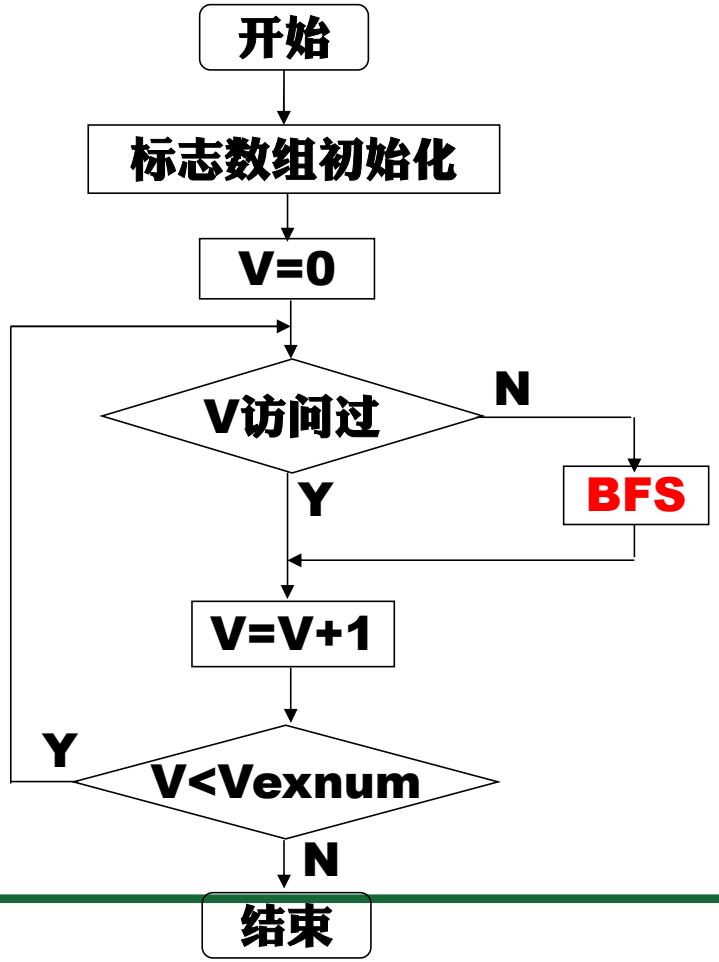
例：



访问次序　a　c　d　e　f　h　k

6.3 图的遍历

图的广度遍历（BFS）



6.3 图的遍历

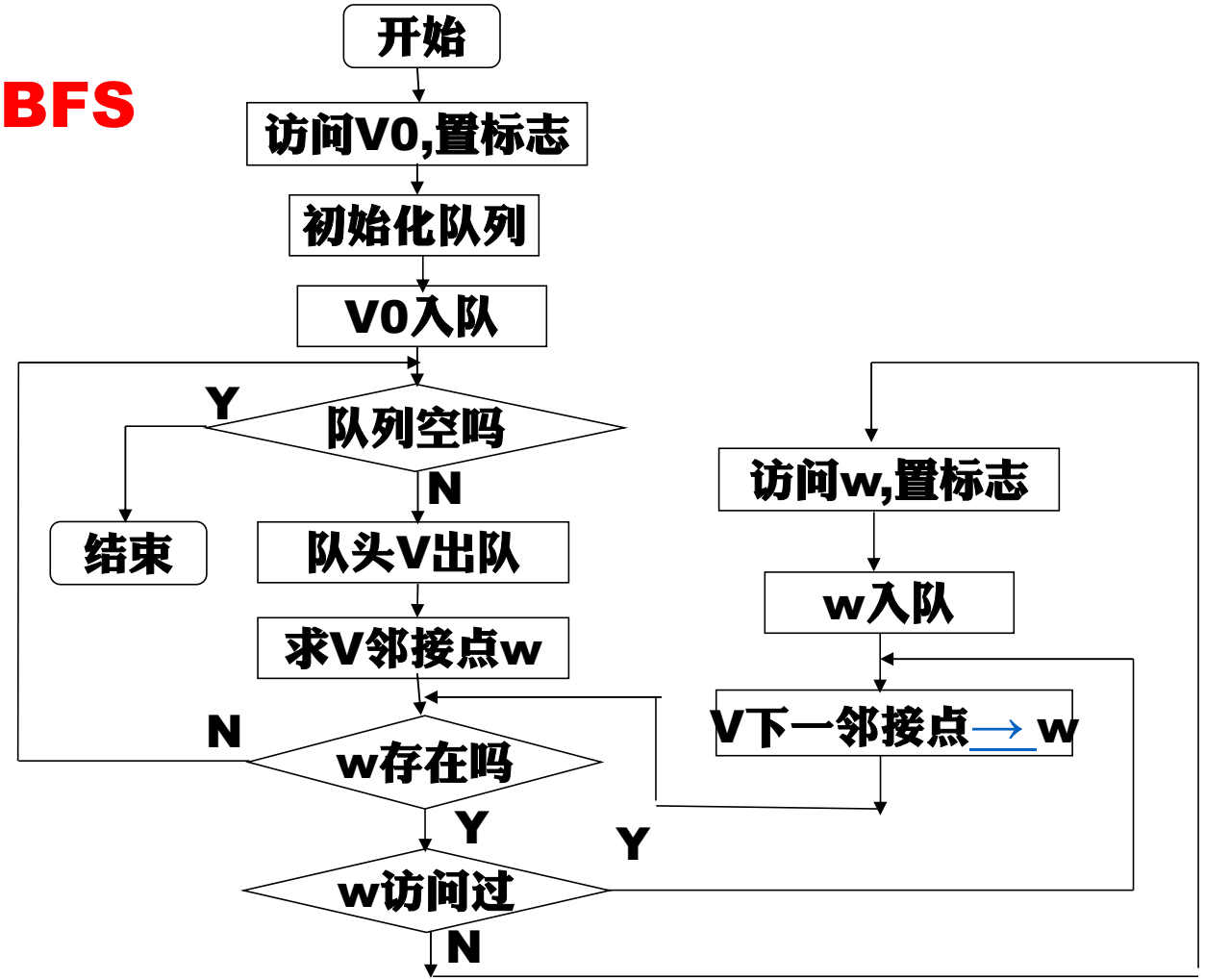
图的广度遍历（BFS）

```
void BFSTraverse ( Graph G, void (* Visit) ( VertexType ) )
{
   //本算法对图G进行广度优先遍历
   for ( v=0; v<G.vexnum; ++v )
        visited[v] = FALSE; // 访问标志数组初始化
   for ( v=0; v<G.vexnum; ++v )
        if ( ! visited[v] )
            BFS( G, v, Visit );
}  //BFSTraverse
```

6.3 图的遍历

图的广度遍历（BFS）——算法6.6

BFS



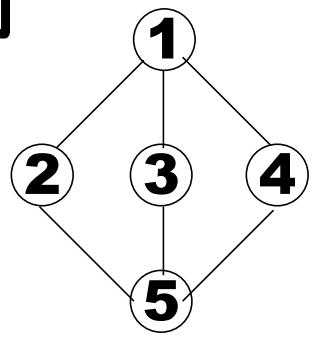
6.3 图的遍历

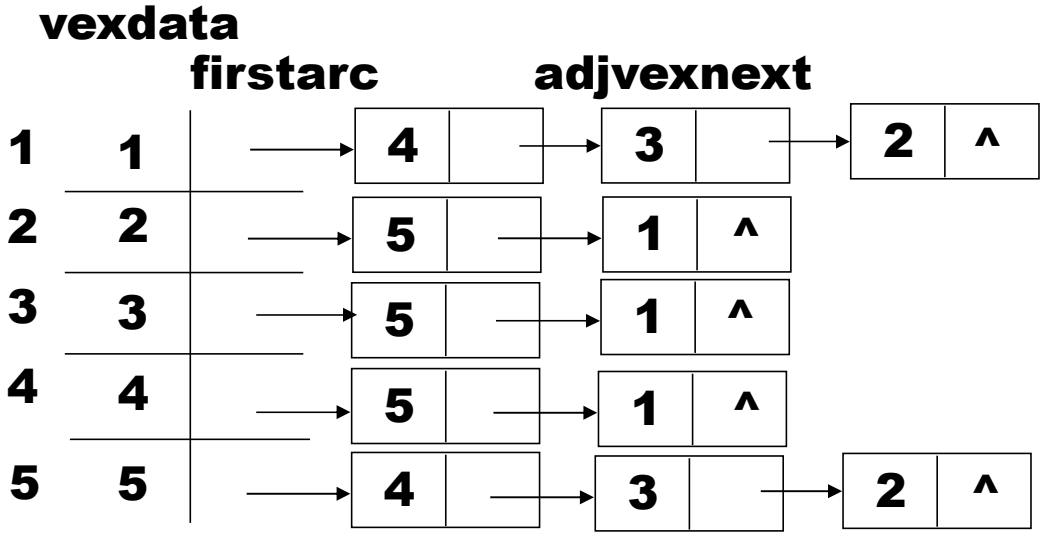
```
void BFS( Graph G, int v, void(* Visit) (VertexType e) )
{ // 从第v个顶点出发
   InitQueue(Q);  // 建立辅助空队列Q
   Visit(v);   visited[v]=TRUE;   // 访问u，访问标志数组
   EnQueue(Q,v);    // v入队
   while ( ! QueueEmpty( Q ) )
   { DeQueue(Q,u);   // 队头元素出队，并赋值给u
     for ( w=FirstAdjVex(G,u); w; w=NextAdjVex(G,u,w) )
          if ( ! visited[w] )
          {   Visit(w);
              visited[w]=TRUE;    // 访问u
              EnQueue(Q,w);
          }
   } //while
} //BFS
```

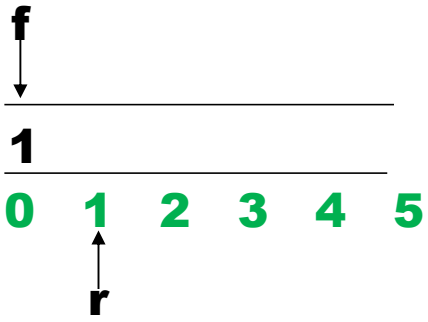
6.3 图的遍历

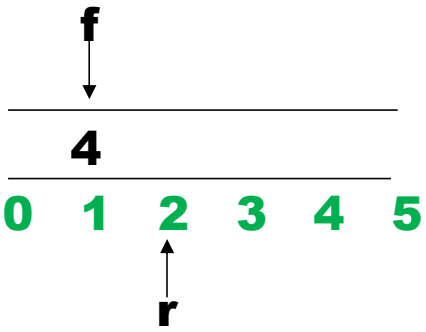
图的广度遍历（BFS）

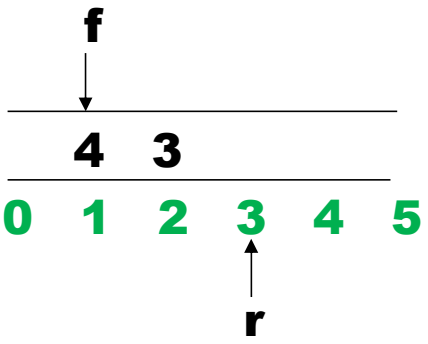
例







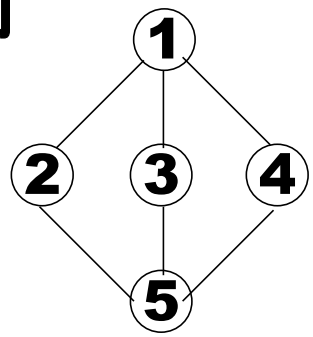


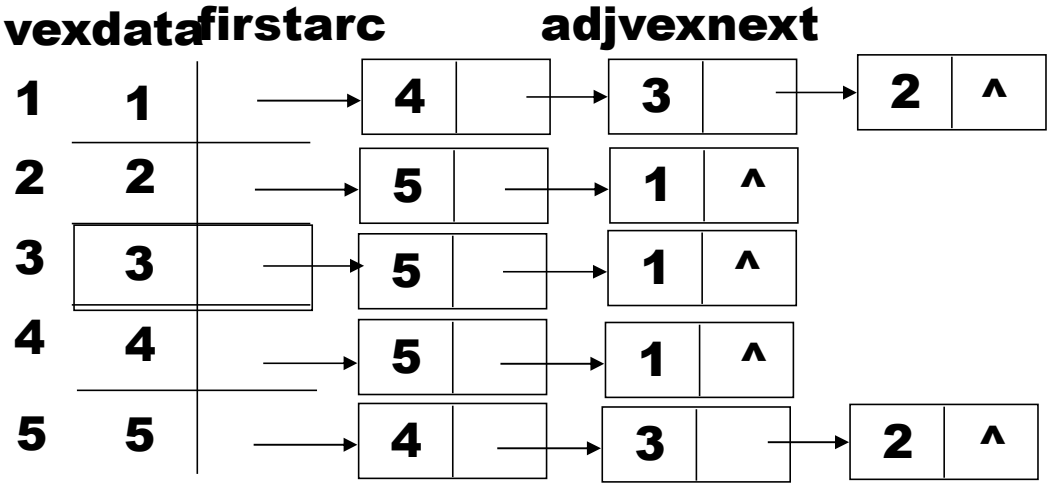


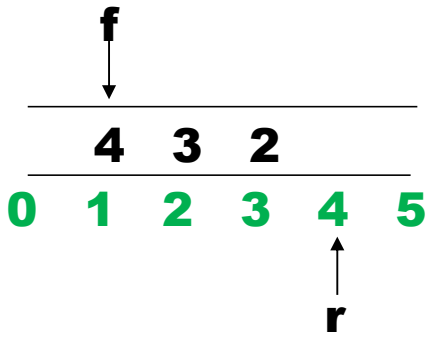
6.3 图的遍历

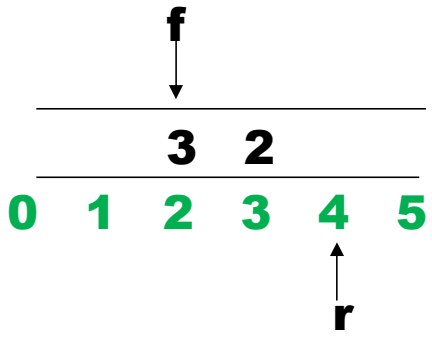
图的广度遍历（BFS）

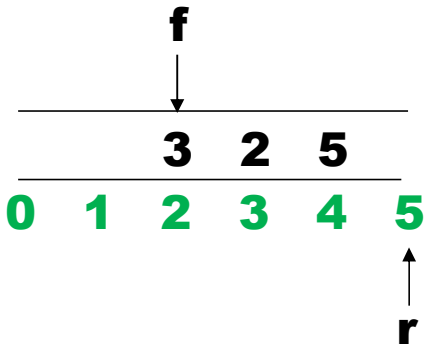
例







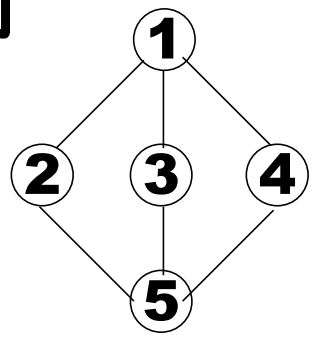


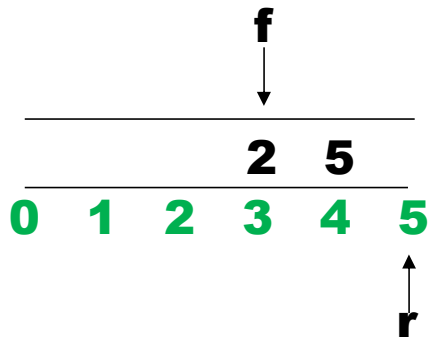


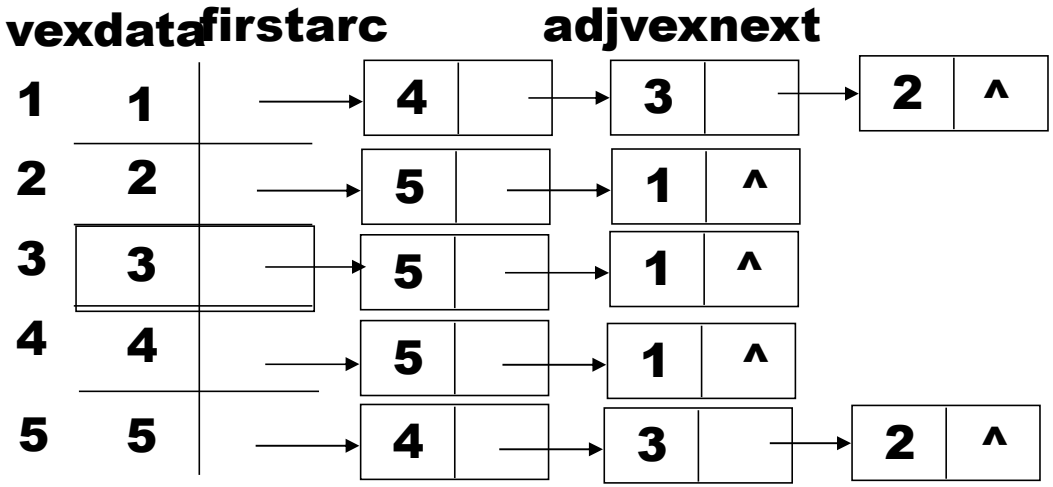
6.3 图的遍历

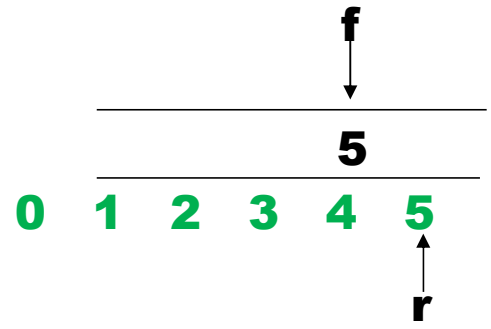
图的广度遍历（BFS）

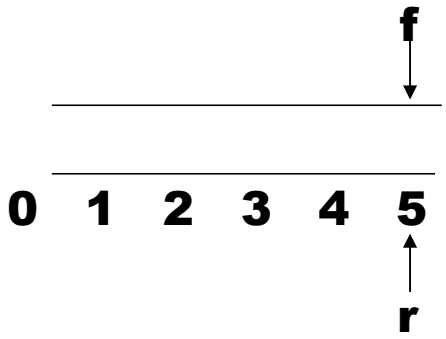
例











6.3 图的遍历

遍历的应用

求两个顶点之间的最短路径长度

广度优先搜索访问顶点的次序是按“路径长度”渐增的次序。

求路径长度最短的路径可以基于广度优先搜索遍历进行。

6.4 图的最小生成树

问题提出

要在n个城市间建立通信联络网，如何省钱?

顶点——表示城市

权——城市间建立通信线路所需花费代价

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

MST(Minimum cost Spanning Tree)

网络中的SpaningTree Protocol

6.4 图的最小生成树

利用 MST 性质构造最小生成树

若U集是V的一个非空子集，若(u0, v0)是一条最小权值的边，其中u0∈U，v0∈V-U；则：(u0, v0)必在最小生成树上。

典型算法

- **普里姆(Prim)算法**

将顶点归并，与边数无关，适于稠密网。

- **克鲁斯卡尔(Kruskal)算法**

将边归并，适于求稀疏网的最小生成树。

6.4 图的最小生成树

普里姆算法（Prim）

设 G=(V, GE) 为一个具有 n 个顶点的连通网络，T=(U, TE) 为构造的生成树。

(1) 初始时，U = {u_0}，TE = ϕ；

(2) 在所有 u $\in$ U 且 v $\in$ V-U 的边 (u, v) 中选择一条权值最小的边，不妨设为(u,v)；

(3) (u,v) 加入TE，同时将 v 加入U；

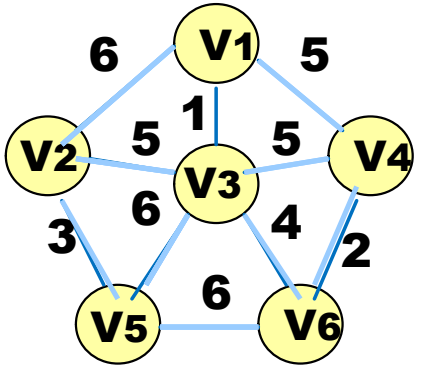
(4) 重复(2)(3)，直到 U=V 为止；

U　　V-U　　U扩大　　V-U缩小

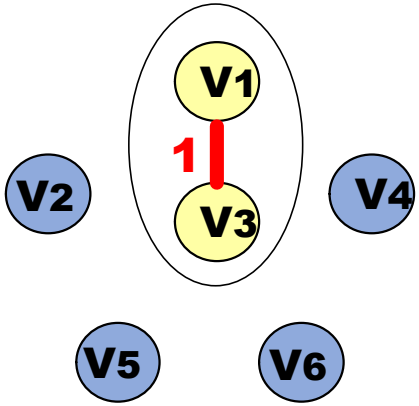
6.4 图的最小生成树

普里姆算法（Prim）——教材P175

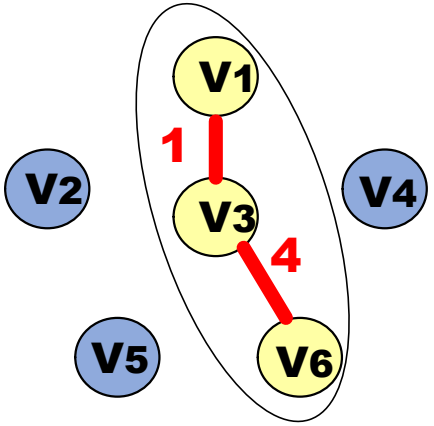
U={ V1 }



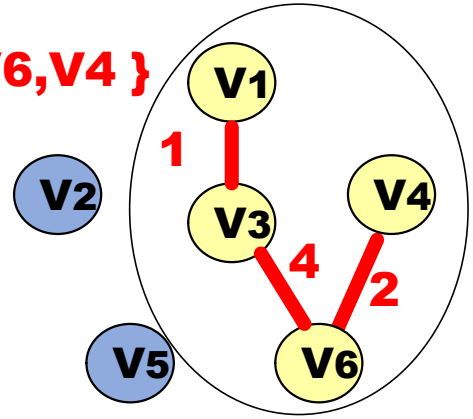
U={ V1,V3 }

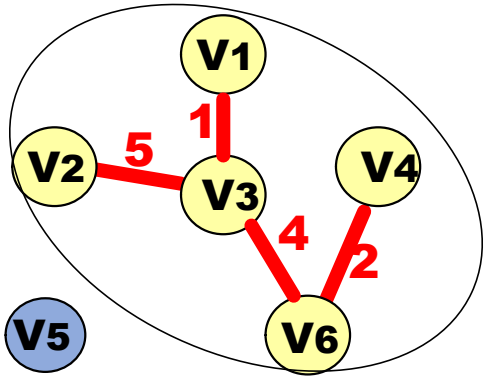


U={ V1,V3,V6}

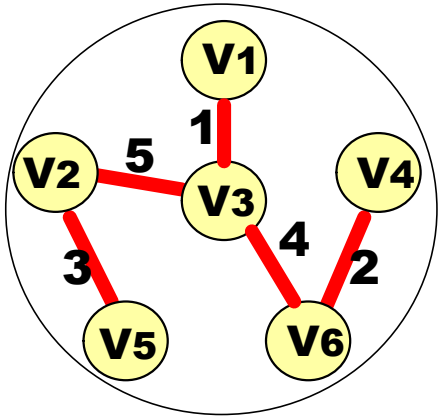


U={ V1,V3,V6,V4 }





U={ V1,V3,V6,V4,V2 }



U={ V1,V3,V6,V4,V2,V5 }

6.4 图的最小生成树

辅助数组closedge[ ]

```
struct {
    VertexType  Adjvex;    // 相关顶点
    VRType      lowcost;   // 最小边的权值
} closedge[ MAX_VERTEX_NUM ];
```

Closedge. Adjvex[v]:

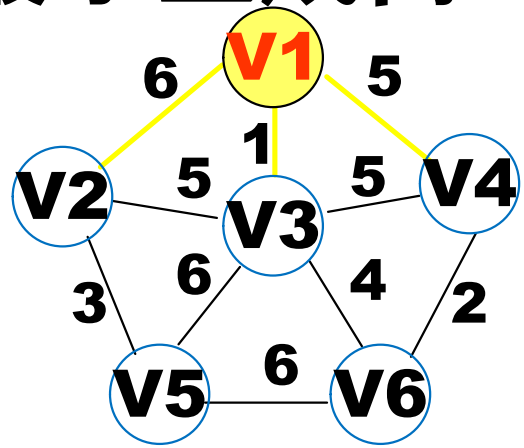
顶点v到子集U中权最小边 (v, u) 关联的顶点u

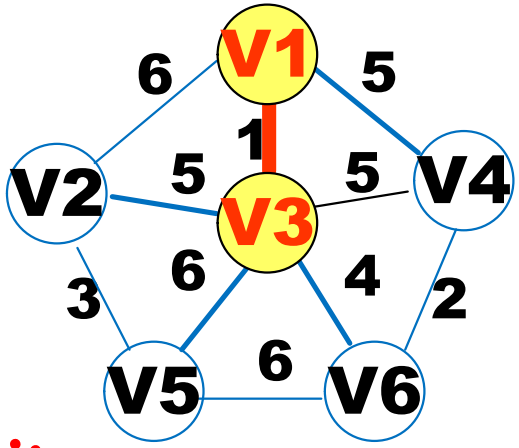
Closedge.lowcost[v]:

顶点v到子集U权最小边 (v, u) 的权值(距离)

	1	2	3	4	5	6
closedge.Adjvex						
closedge.Lowcost						

6.4 图的最小生成树

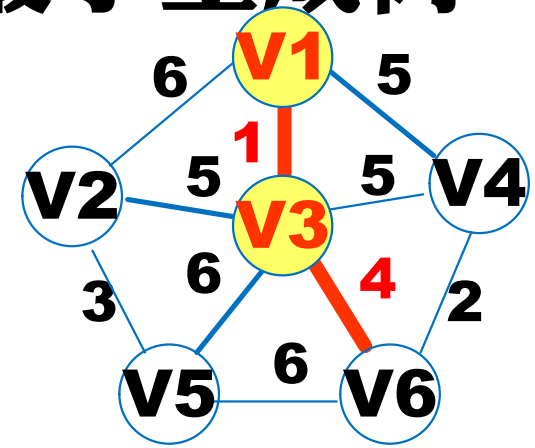


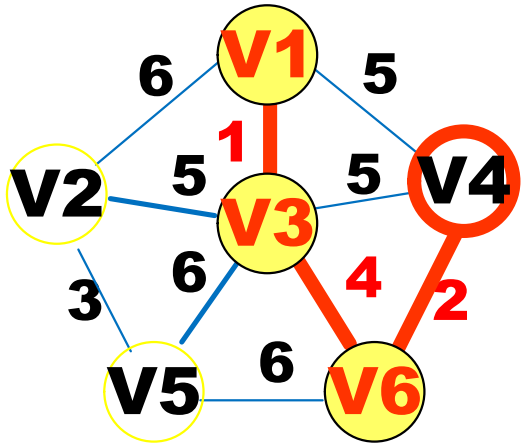


	0(V1)	1(V2)	2(V3)	3(V4)	4(V5)	5(V6)
closedge.Adjvex	V1	V1	V1	V1	V1	V1
closedge.Lowcost	0	6	1	5	max	max

	0(V1)	1(V2)	2(V3)	3(V4)	4(V5)	5(V6)
closedge.Adjvex	V1	V3	V1	V1	V3	V3
closedge.Lowcost	0	5	1	5	6	4

6.4 图的最小生成树

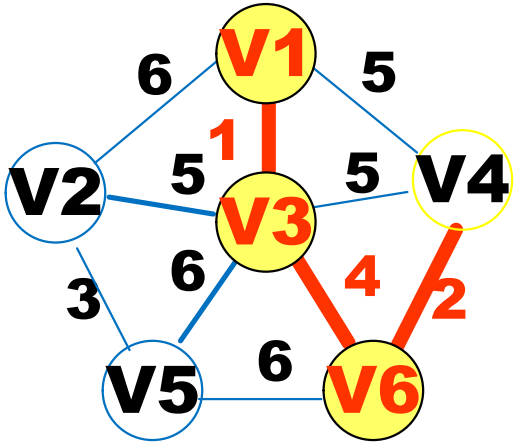


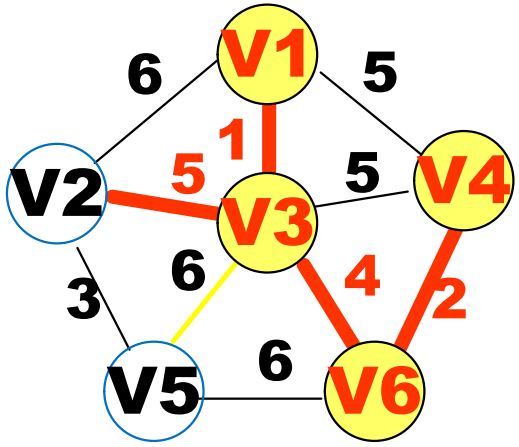


	0(V1)	1(V2)	2(V3)	3(V4)	4(V5)	5(V6)
closedge.Adjvex		V3	V1	V1	V3	V3
closedge.Lowcost	0	5	1	5	6	4

	0(V1)	1(V2)	2(V3)	3(V4)	4(V5)	5(V6)
closedge.Adjvex		V3	V1	V6	V3	V3
closedge.Lowcost	0	5	1	2	6	4

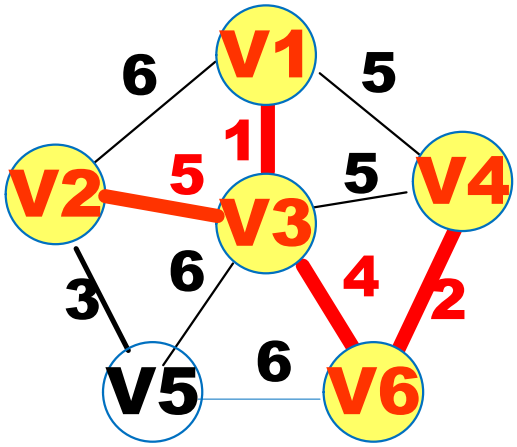
6.4 图的最小生成树

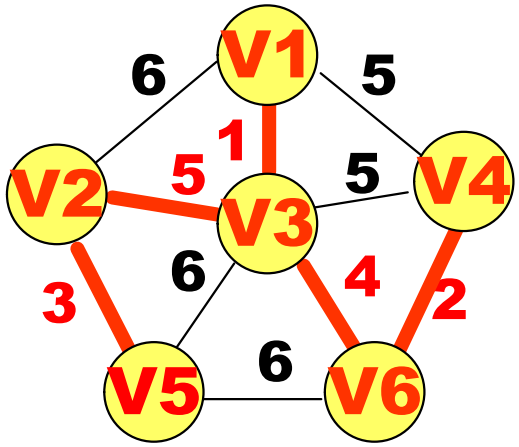




	0(V1)	1(V2)	2(V3)	3(V4)	4(V5)	5(V6)
closedge.Adjvex		V3	V1	V6	V3	V3
closedge.Lowcost	0	5	1	2	6	4

6.4 图的最小生成树





	0(V1)	1(V2)	2(V3)	3(V4)	4(V5)	5(V6)
closedge.Adjvex		V3	V1	V6	V3	V3
closedge.Lowcost	0	5	1	2	3	4

6.4 图的最小生成树

```
void MiniSpanTree_P( MGraph G, VertexType u )
{
//用普里姆算法从顶点u出发构造网G的最小生成树
 k = LocateVex ( G, u );
 for ( j=0; j<G.vexnum; ++j )  // 辅助数组初始化
     if (j!=k)
         closedge[j] = { u, G.arcs[k][j] };
 closedge[k].Lowcost = 0;       // 初始，U={u}
 for ( i=0; i<G.vexnum; ++i )
 {

      继续向生成树上添加顶点;

 }
```

6.4 图的最小生成树

```
k = minimum(closedge);
                    // 求出加入生成树的下一个顶点(k)
printf(closedge[k].Adjvex, G.vexs[k]);
                    // 输出生成树上一条边
closedge[k].Lowcost = 0;     // 第k顶点并入U集
for (j=0; j<G.vexnum; ++j) //修改其它顶点的最小边
  if ( G.arcs[k][j] < closedge[j].Lowcost )
     closedge[j] = { G.vexs[k], G.arcs[k][j] };
```

6.4 图的最小生成树

普里姆算法的性能

设n是图的顶点数，普里姆算法的时间复杂度为$O(n^2)$。

与边数无关，适用于求**边稠密**的网的最小生成树。

6.4 图的最小生成树

克鲁斯卡尔(Kruskal)算法

设连通网 N = (V, {E})。

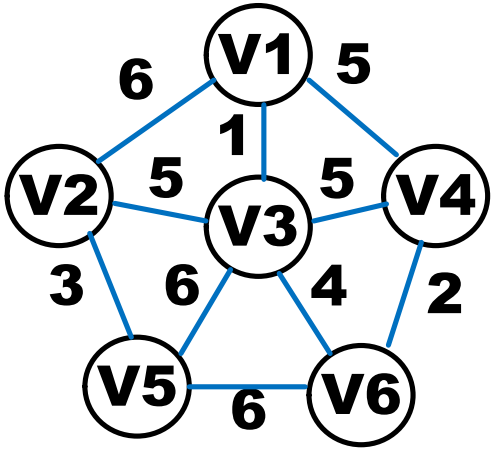
1) 初始时最小生成树只包含图的n个顶点，每个顶点为一棵子树；

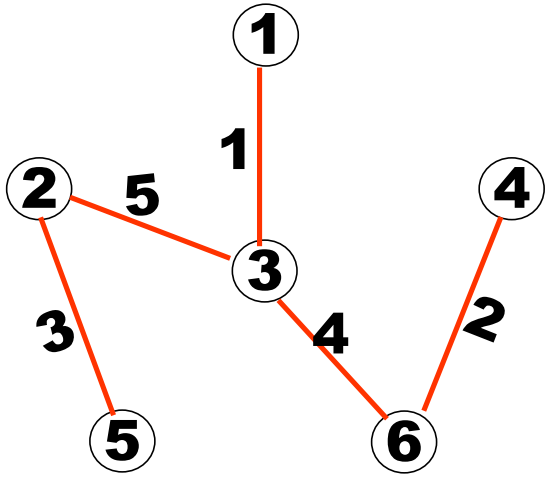
2) 选取权值较小且所关联的两个顶点不在同一子树的边，将此边加入到最小生成树中；

3) 重复2）n-1次，即得到包含n个顶点和n-1条边的最小生成树。

6.4 图的最小生成树

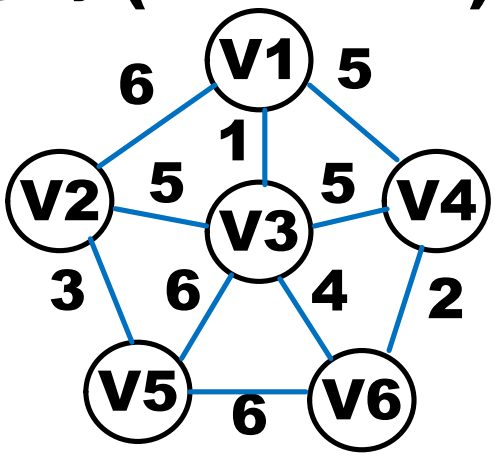
克鲁斯卡尔(Kruskal)算法

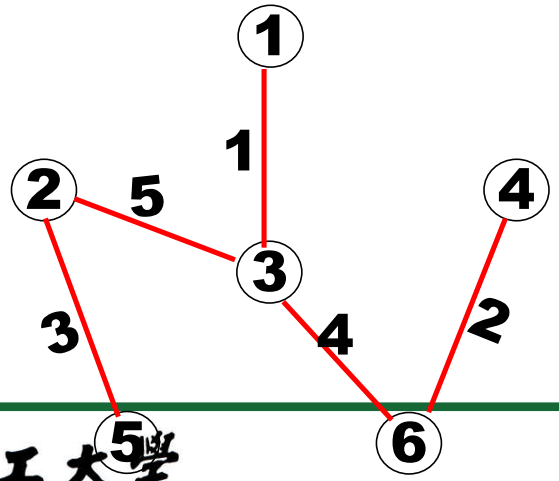




6.4 图的最小生成树

克鲁斯卡尔(Kruskal)算法





	data	jihe
1	1	2
2	2	2
3	3	1 2
4	4	1 2
5	5	2
6	6	4 1 2

采用边集数组的形式保存图：

	vexh	vext	weight	flag
0	1	2	6	0
1	1	3	1	1
2	1	4	5	2
3	2	3	5	1
4	2	5	3	1
5	3	4	5	0
6	3	5	6	0
7	3	6	4	1
8	4	6	2	1
9	5	6	6	0

6.4 图的最小生成树

克鲁斯卡尔的性能

设图的边数是e，克鲁斯卡尔算法的时间复杂度为O(elog e)。

适用于求边稀疏的网的最小生成树。

6.4 图的最小生成树

两种算法比较

	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

6.5 有向无环图——拓扑排序

问题提出：学生选修课程问题

顶点——表示课程

有向弧——表示先决条件，若 课程i 是 课程j 的先决条件，则图中有弧<i,j>。

学生应按怎样的顺序学习这些课程，才能无矛盾、顺利地完成学业——拓扑排序。

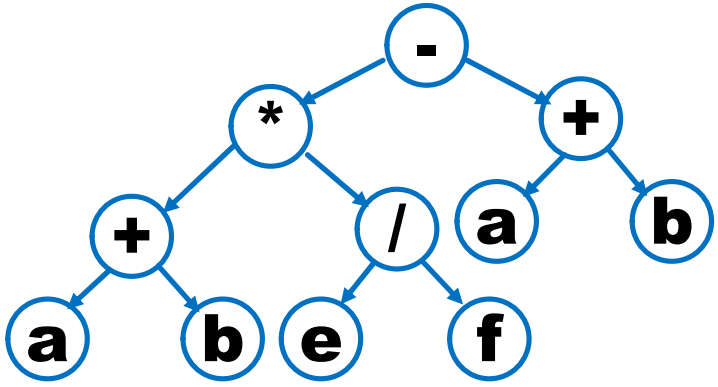
6.5 有向无环图——拓扑排序

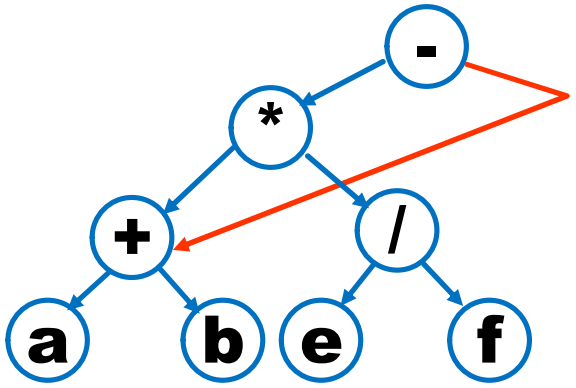
有向无环图(DAG)

没有回路的有向图。

含有公共子式的表达式

(a + b) * (e / f) - (a + b)

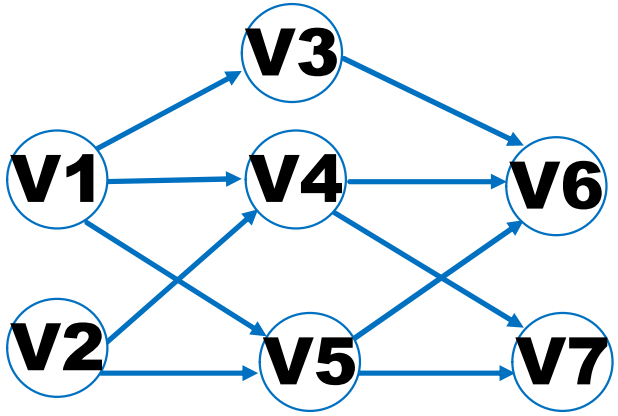




6.5 有向无环图——拓扑排序

有向无环图(DAG)

某工程可分为7个子工程，工程流程图。



6.5 有向无环图——拓扑排序

定义

AOV网——用顶点表示活动，用弧表示活动间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。

若 $<v_i,v_j>$ 是图中有向边，则 v_i 是 v_j 的直接前驱；v_j 是 v_i 的直接后继。

AOV网中不允许有回路，这意味着某项活动以自己为先决条件。

6.5 有向无环图——拓扑排序

拓扑排序

把AOV网络中各顶点按照它们相互之间的优先关系排列成一个线性序列的过程。

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。

6.5 有向无环图——拓扑排序

拓扑排序的方法

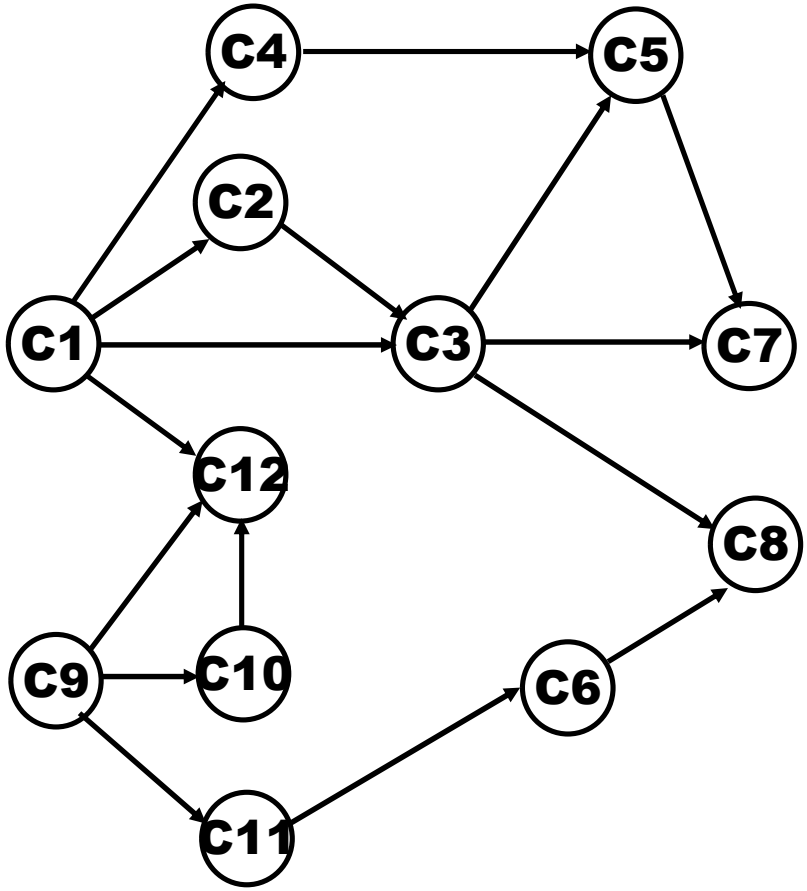
在有向图中选一个没有前驱的顶点且输出。

从图中删除该顶点和所有以它为尾的弧。

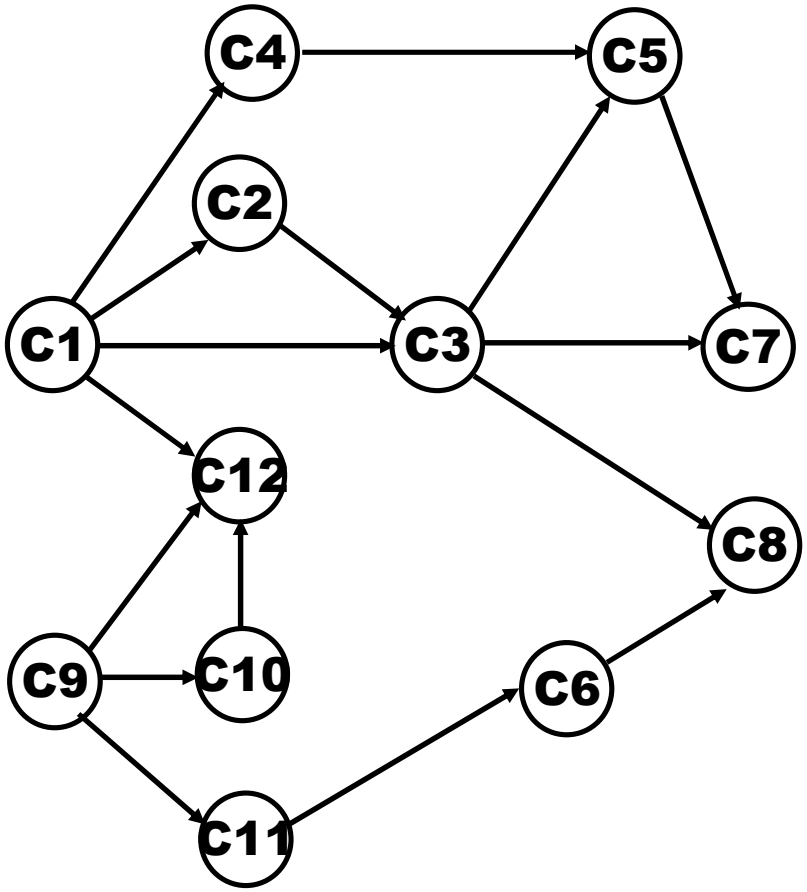
重复上述两步，直至全部顶点均已输出；或者当图中不存在无前驱的顶点为止。

6.5 有向无环图——拓扑排序

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



6.5 有向无环图——拓扑排序

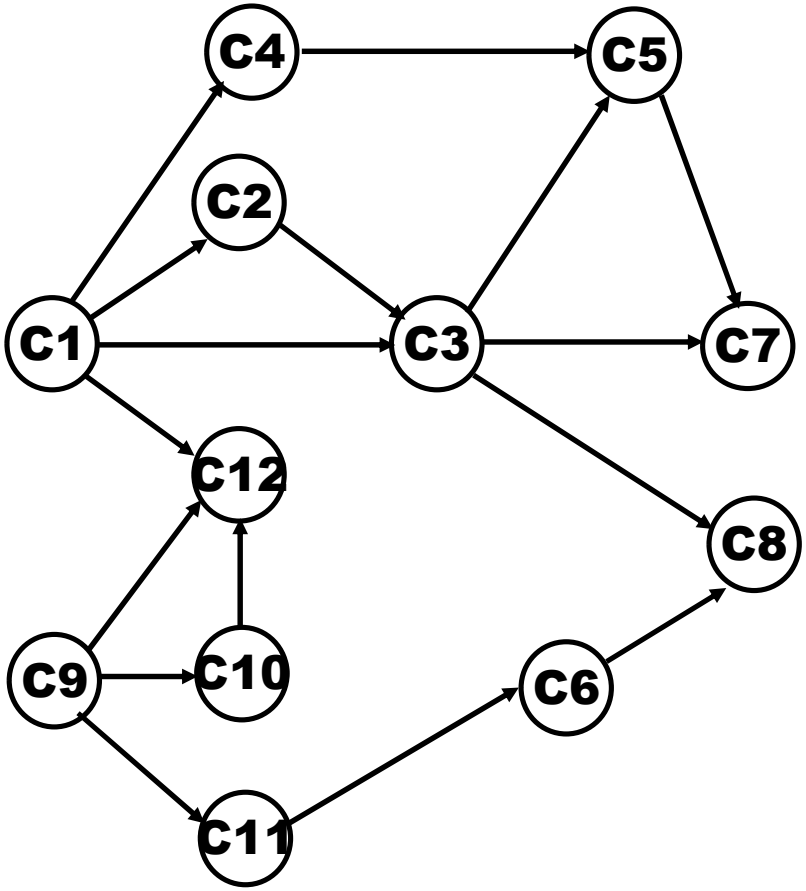


拓扑序列： C1--C2--C3--C4--C5--C7--C9--C10--C11--C6--C12--C8

或： C9--C10--C11--C6--C1--C12--C4--C2--C3--C5--C7--C8

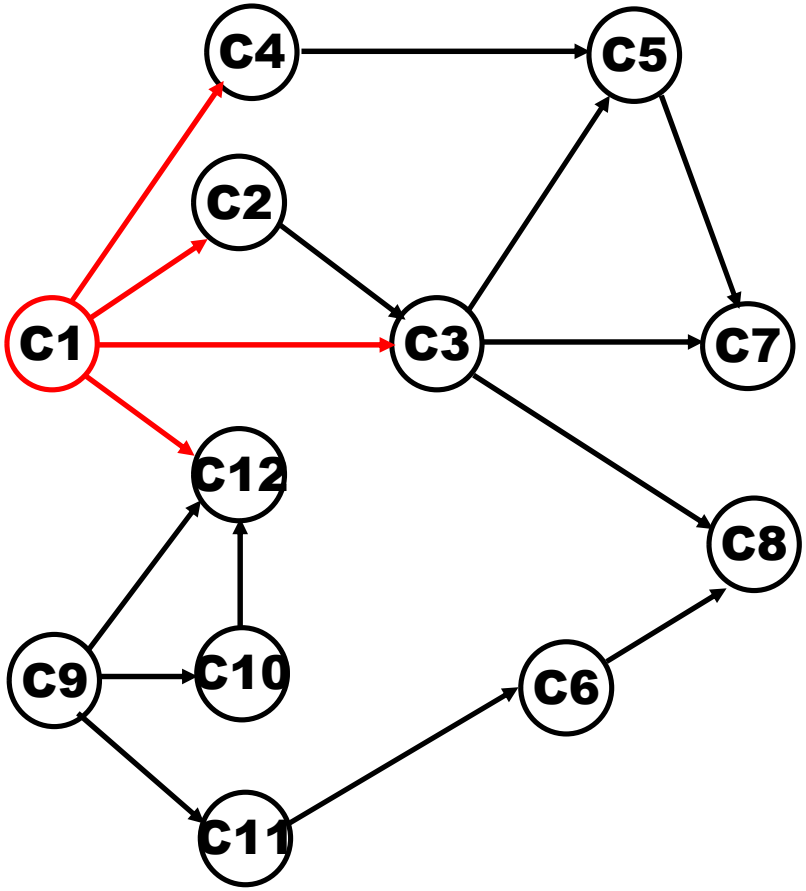
一个AOV网的拓扑序列不是唯一的

6.5 有向无环图——拓扑排序



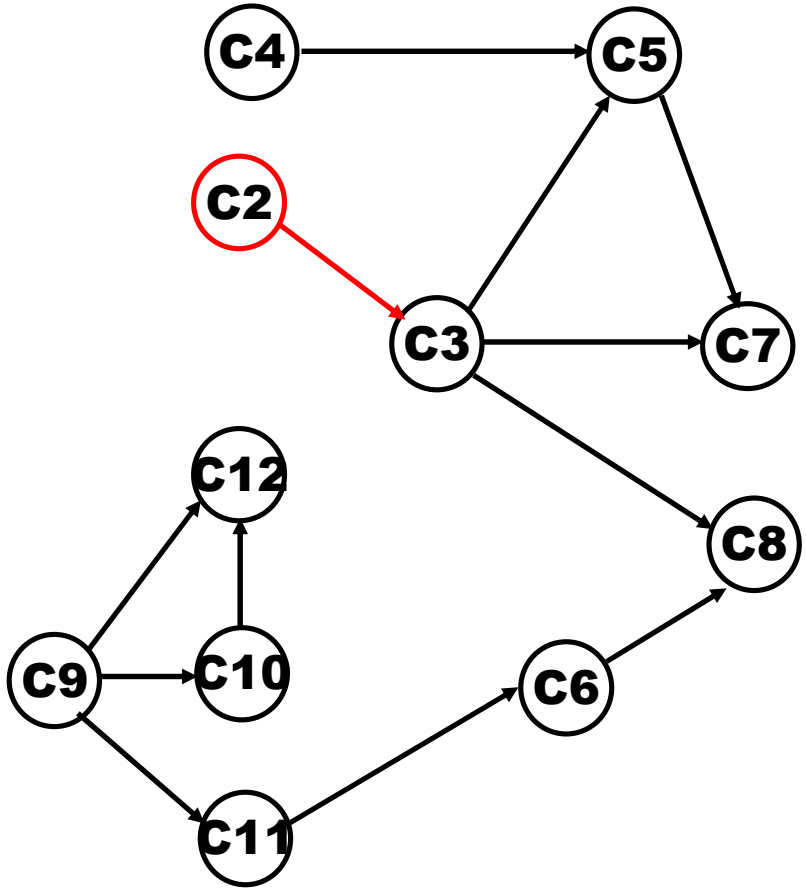
拓扑序列：C1

6.5 有向无环图——拓扑排序



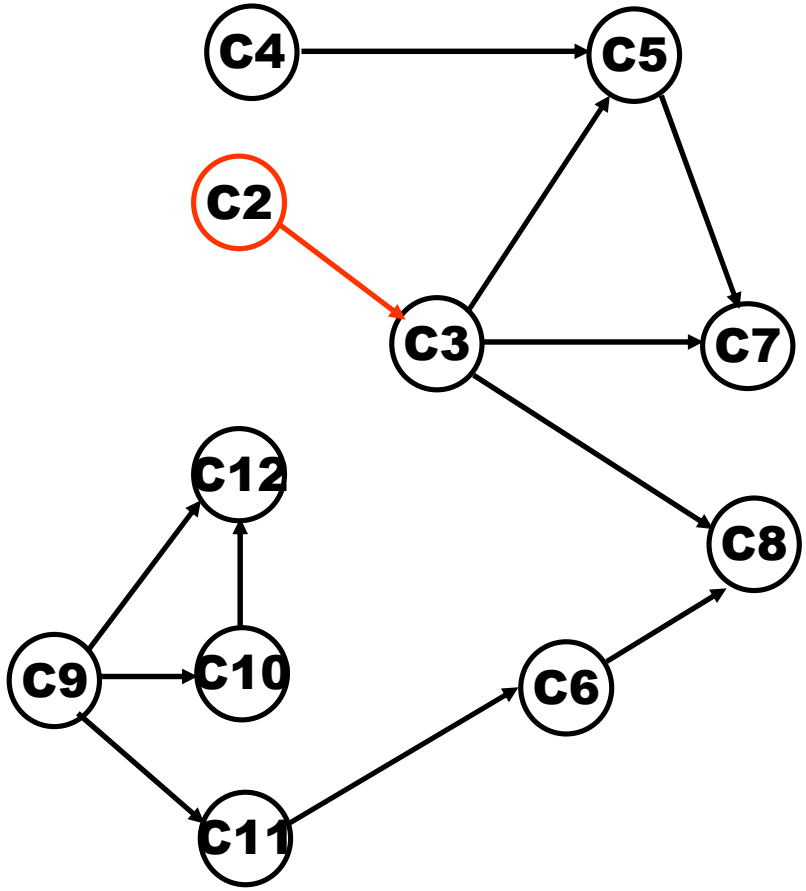
拓扑序列：C1

6.5 有向无环图——拓扑排序



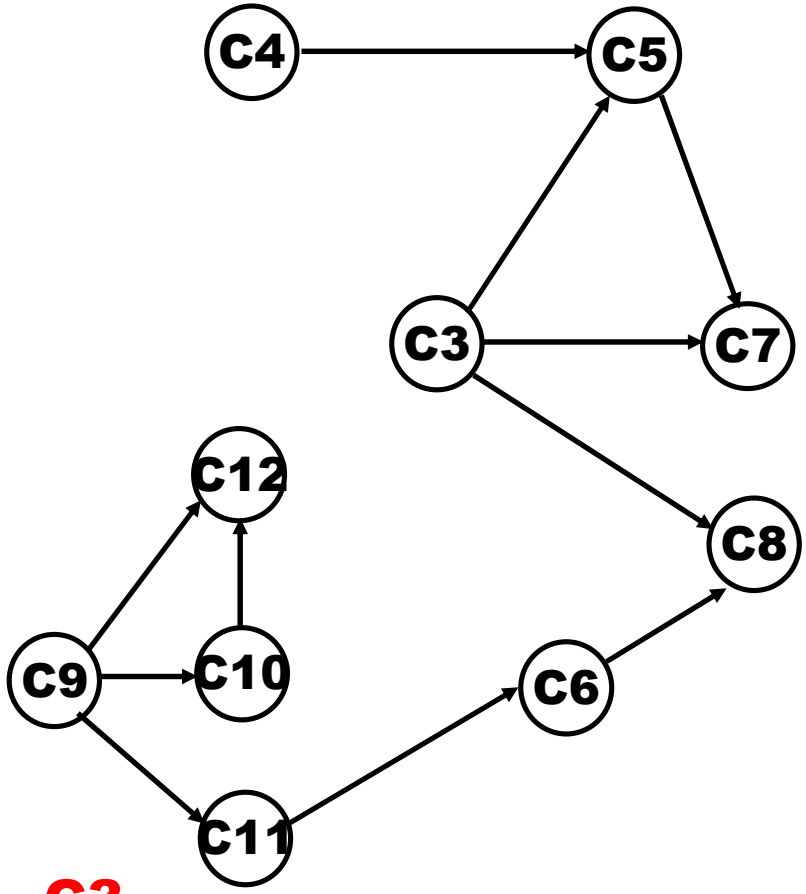
拓扑序列：C1 --C2

6.5 有向无环图——拓扑排序



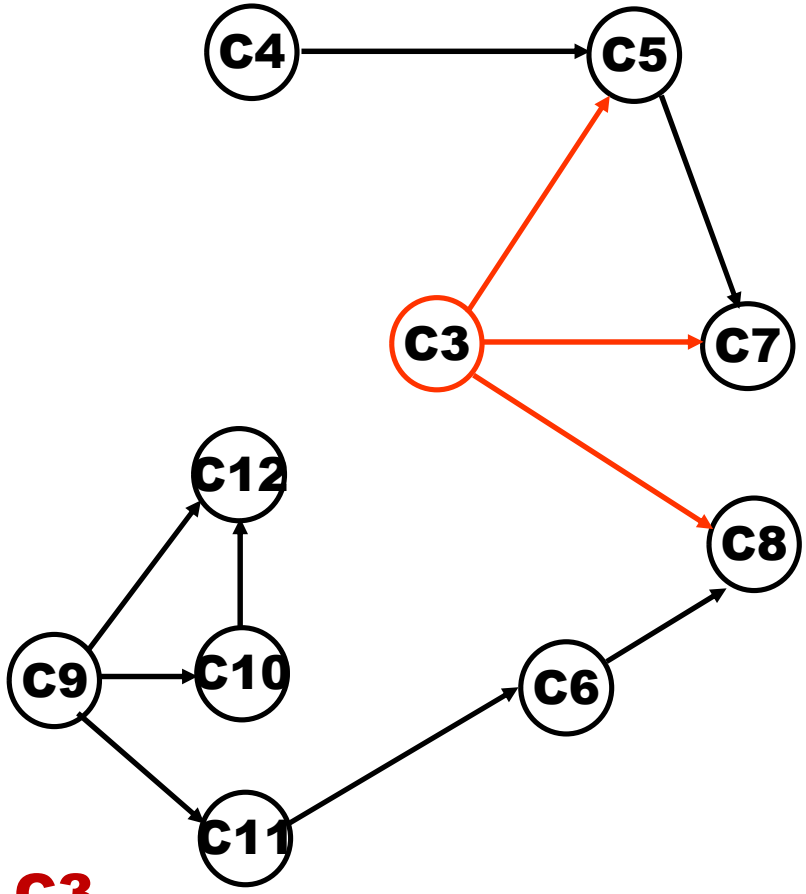
拓扑序列：C1 --C2

6.5 有向无环图——拓扑排序



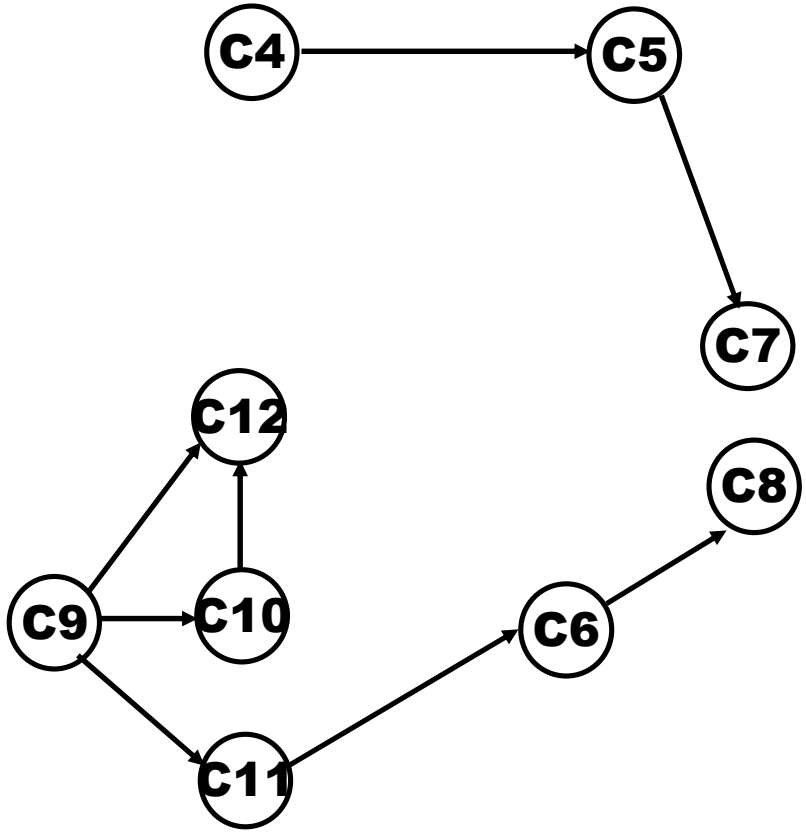
拓扑序列：C1 --C2 --C3

6.5 有向无环图——拓扑排序



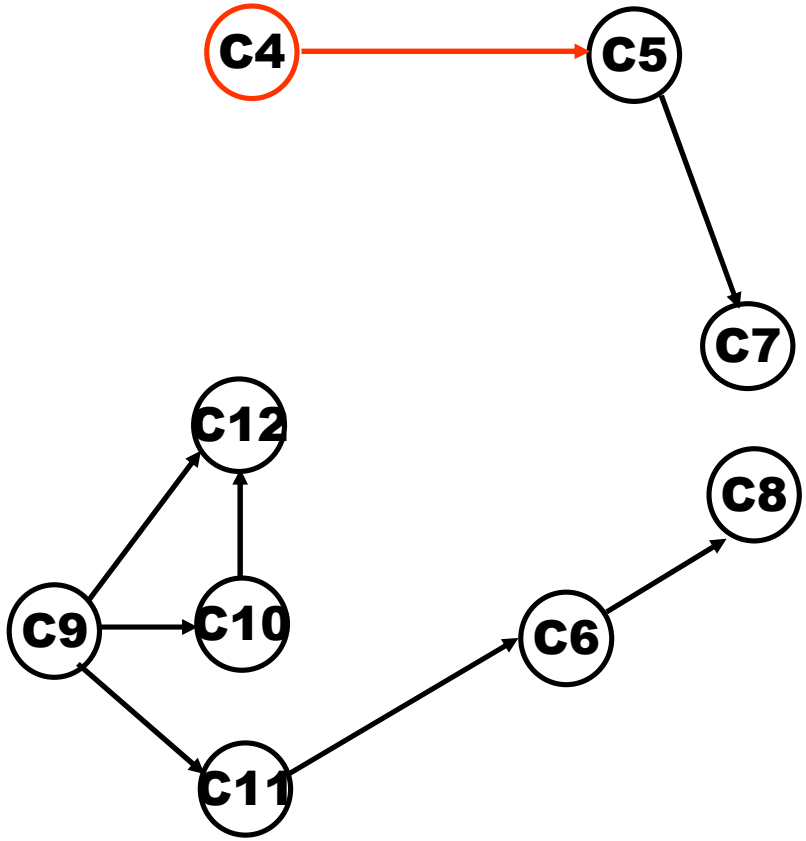
拓扑序列：C1--C2 --C3

6.5 有向无环图——拓扑排序



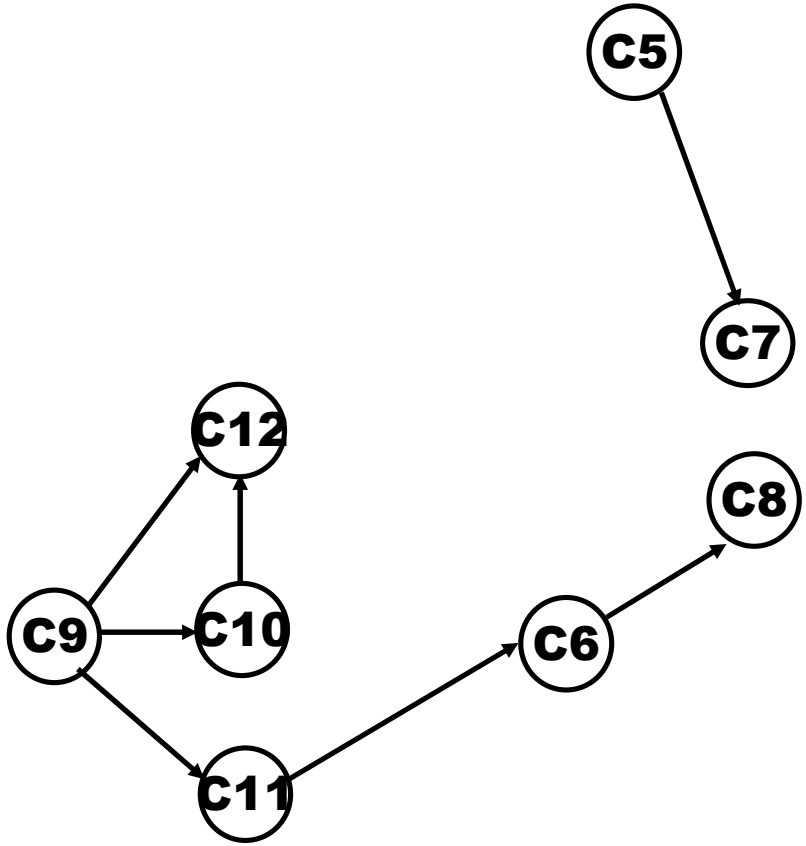
拓扑序列：C1 --C2 --C3 --C4

6.5 有向无环图——拓扑排序



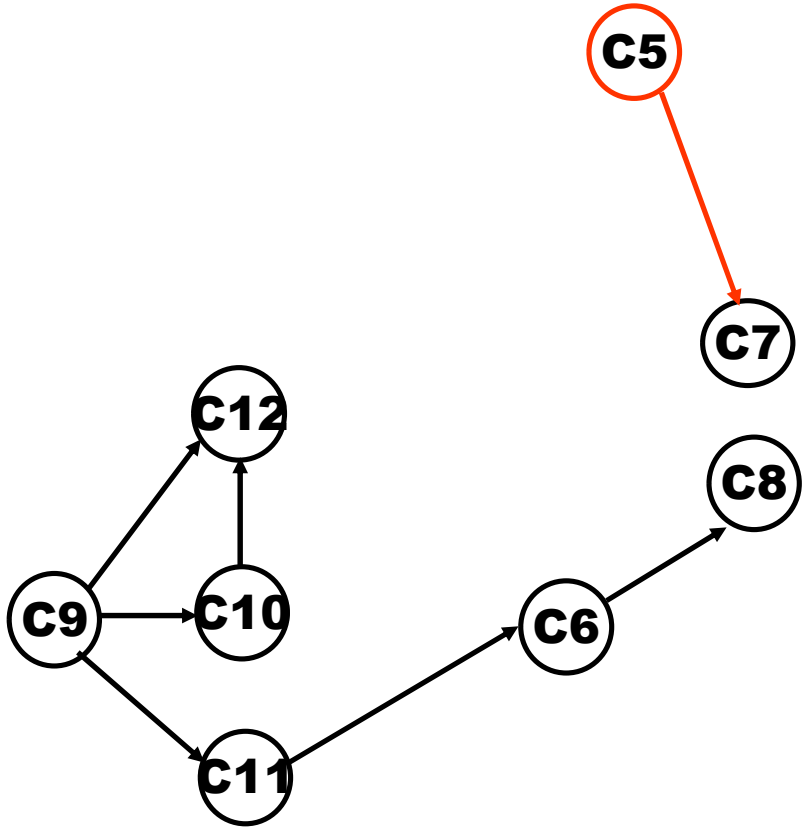
拓扑序列：C1 --C2 --C3 --C4

6.5 有向无环图——拓扑排序



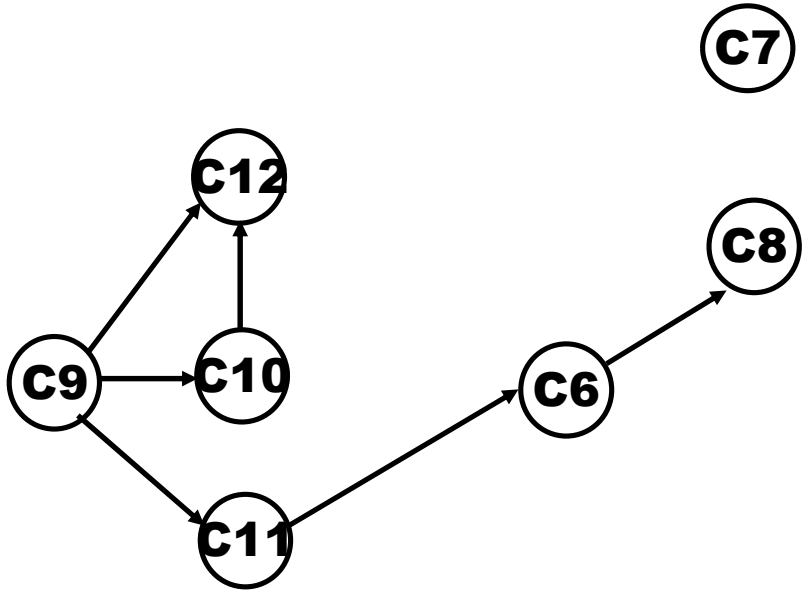
拓扑序列：C1 --C2 --C3 --C4 --C5

6.5 有向无环图——拓扑排序



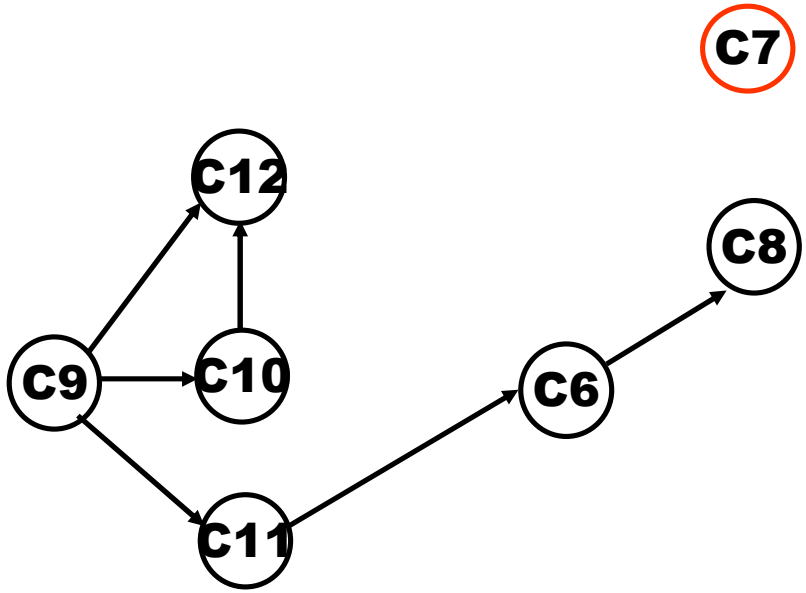
拓扑序列：C1 --C2 --C3 --C4 --C5

6.5 有向无环图——拓扑排序



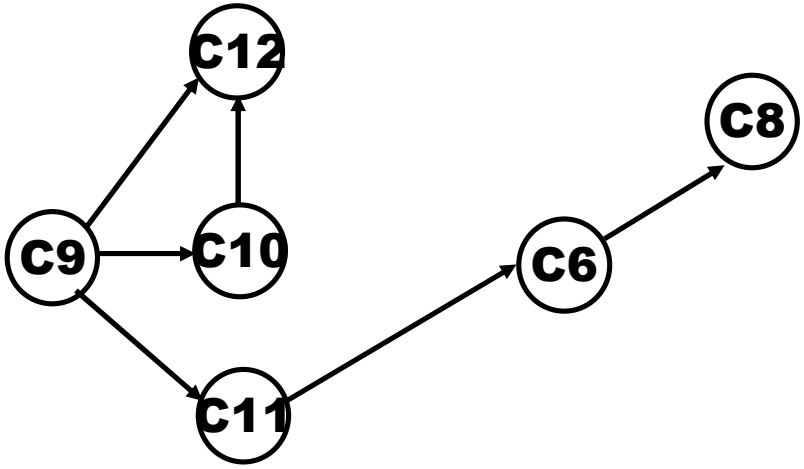
拓扑序列：C1--C2--C3--C4--C5--C7

6.5 有向无环图——拓扑排序



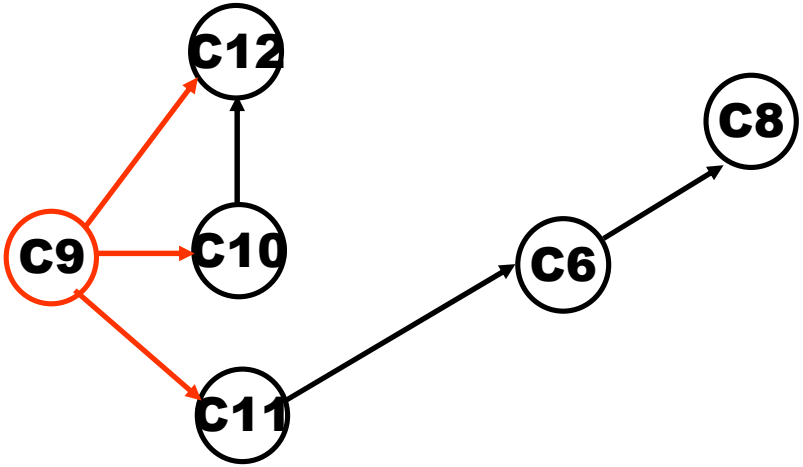
拓扑序列：C1-- -- -- -- --
C2 C3 C4 C5 C7

6.5 有向无环图——拓扑排序



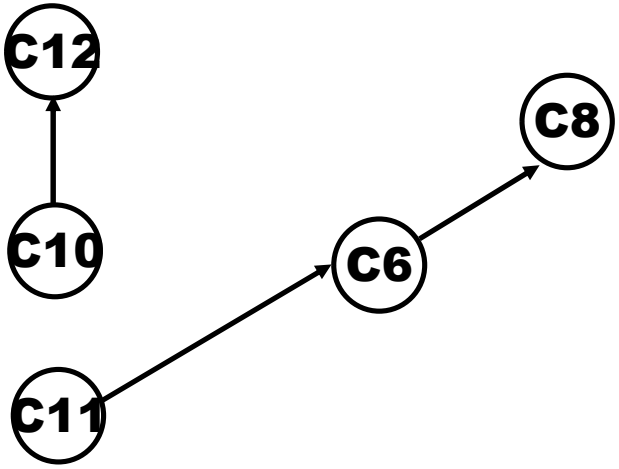
拓扑序列：C1--C2--C3--C4--C5--C7--C9

6.5 有向无环图——拓扑排序



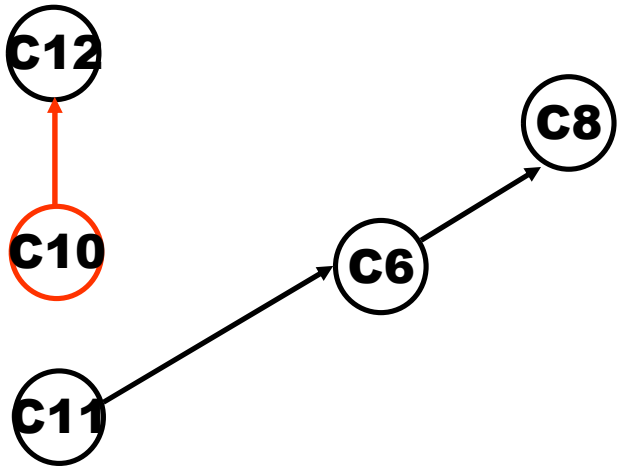
拓扑序列：C1--C2--C3--C4--C5--C7--C9

6.5 有向无环图——拓扑排序



拓扑序列：C1--C2--C3--C4--C5--C7--C9--C10

6.5 有向无环图——拓扑排序



拓扑序列：C1--C2--C3--C4--C5--C7--C9--C10
--C11--C6 --C12 --C8