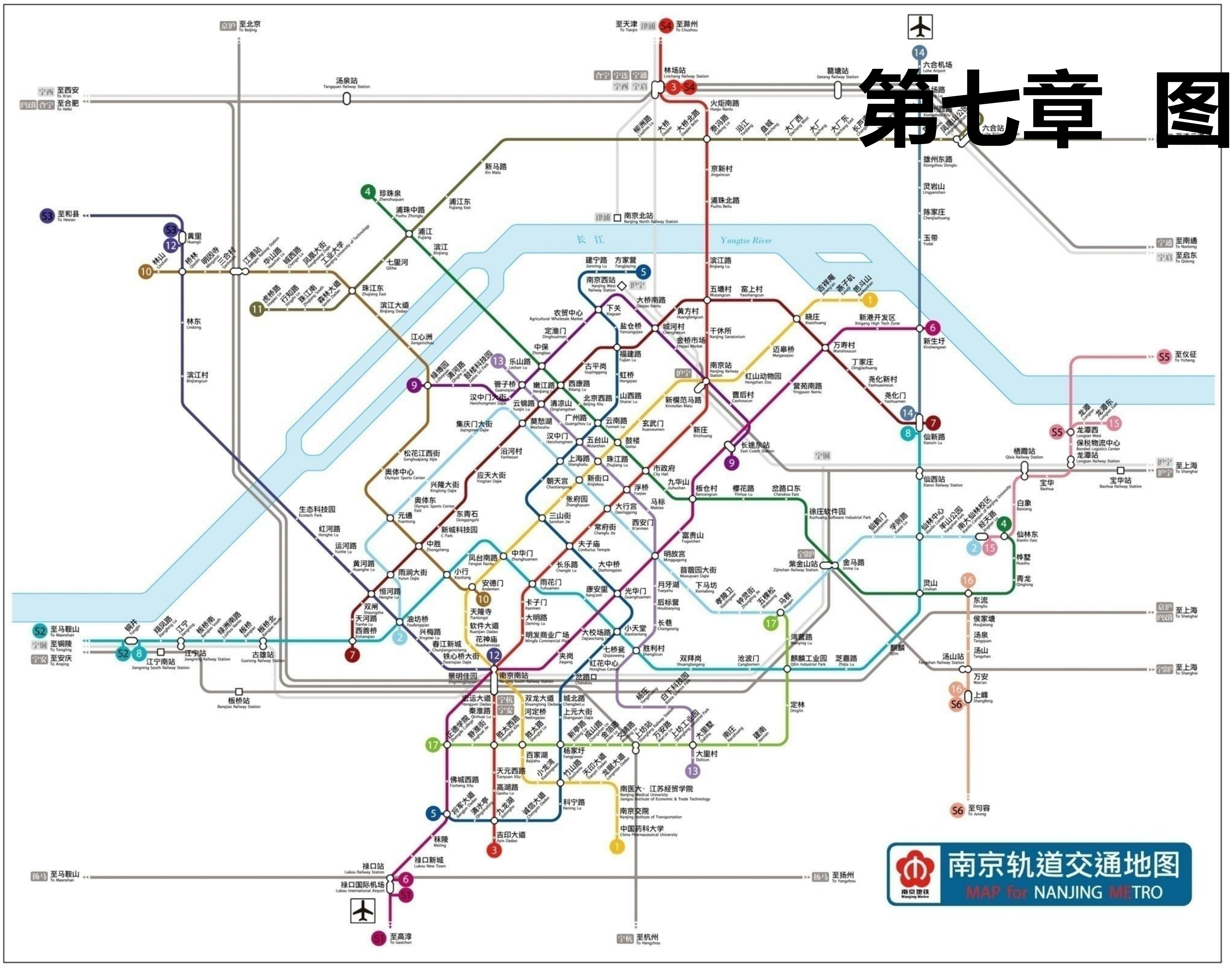
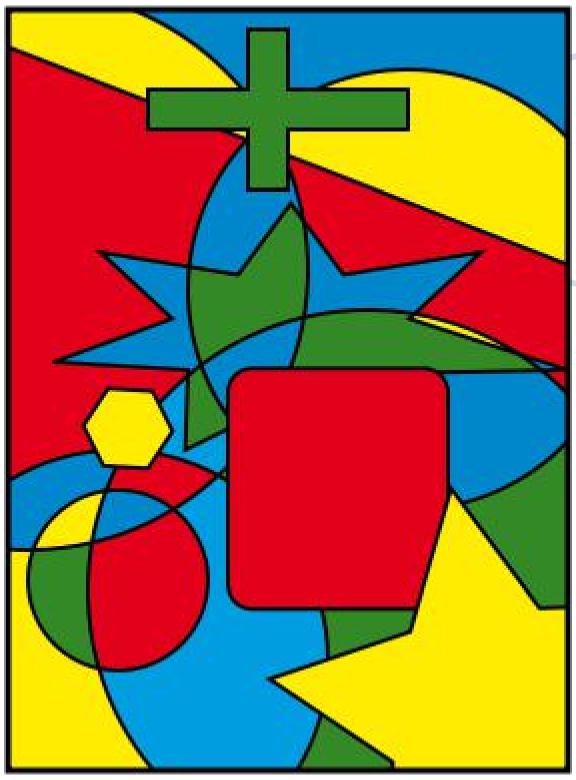
第七章 图

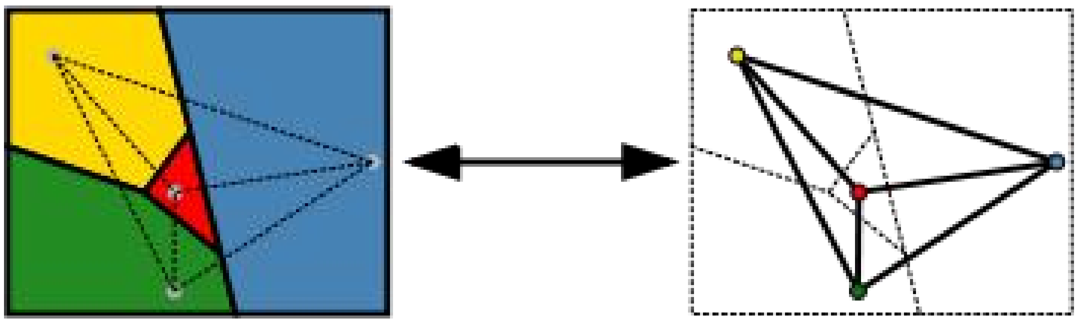




中国地图

审图号：GS(2016)2879号

自然资源部 监制



7.1 抽象数据类型图的定义

7.2 图的存储表示

7.3 图的遍历

7.4 最小生成树

7.5.1 拓扑排序

7.5.2 关键路径

7.6 两点之间的最短路径问题

图的结构定义：

图是由一个顶点集 V 和一个弧集 VR构成的数据结构。

Graph = (V , VR)

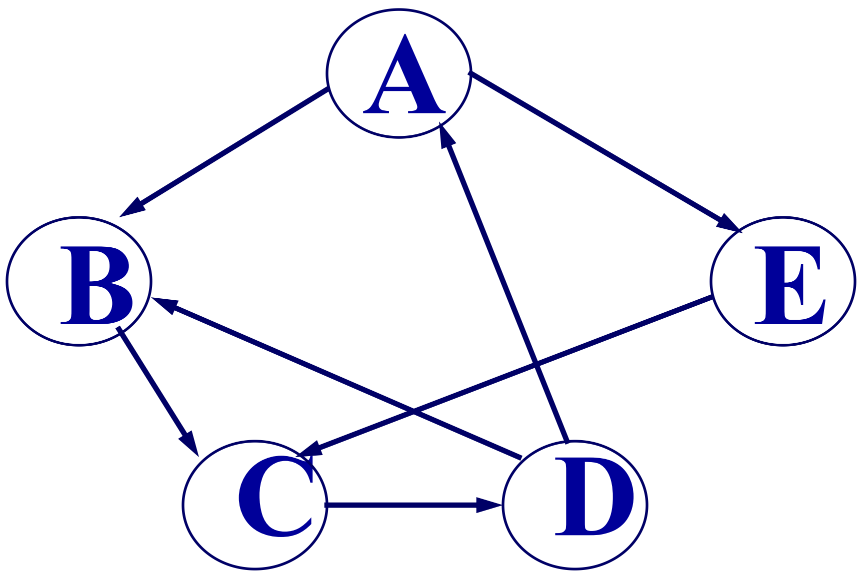
其中， VR= {<v,w>| v,w∈V 且 P(v,w)}

<v,w>表示从 v 到 w 的一条弧，并称 v 为**弧头**， w 为**弧尾**。

谓词 P(v,w) 定义了弧 <v,w>的意义或信息。

由于“弧”是有方向的，因此称由顶点集和弧集构成的图为**有向图**。

例如: $G_1 = (V_1, VR_1)$



其中

$V_1=\{A, B, C, D, E\}$

$VR_1=\{<A,B>, <A,E>, <B,C>, <C,D>, <D,B>, <D,A>, <E,C>\}$

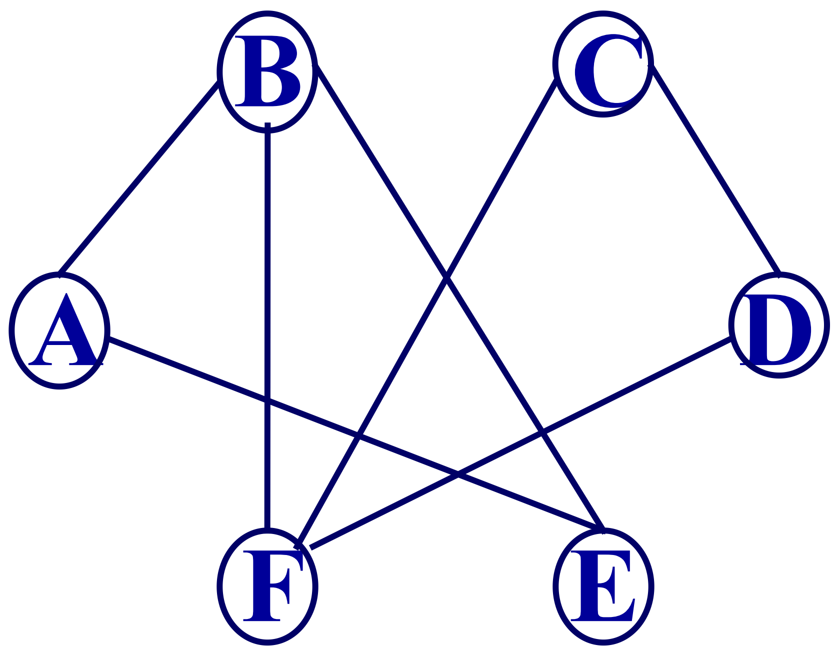
若$<v, w>\in VR$ 必有$<w, v>\in VR$，则称 (v,w) 为顶点 v 和顶点 w 之间存在一条**边**。

由顶点集和边集构成的图称作**无向图**。

例如：$G_2=(V_2,VR_2)$

$V_2=\{A, B, C, D, E, F\}$

$VR_2=\{(A,B), (A,E), (B,E), (C,D), (D,F), (B,F), (C,F)\}$



名词和术语

网、 子图 ➡

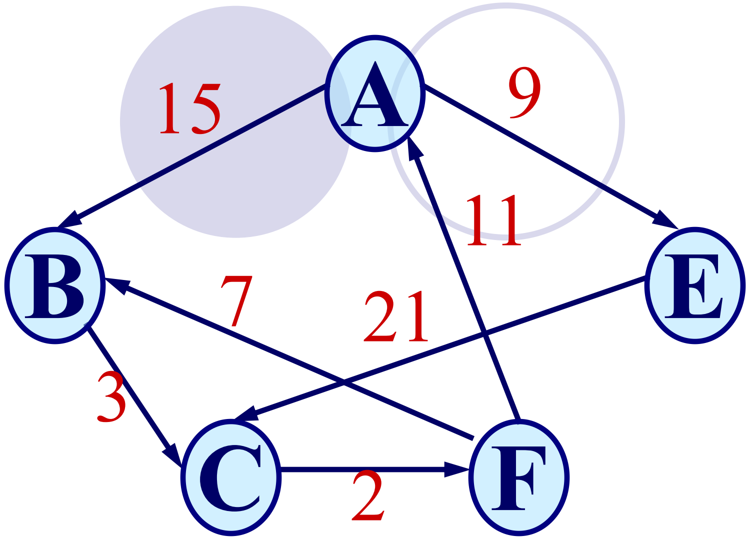
完全图 、稀疏图、 稠密图 ➡

邻接点、 度、 入度、 出 度 ➡

路径、 路径长度、 简 单路径、简 单回 路➡

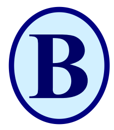
连通图、 连通分量、
强 连通图、 强 连通 分量 ➡

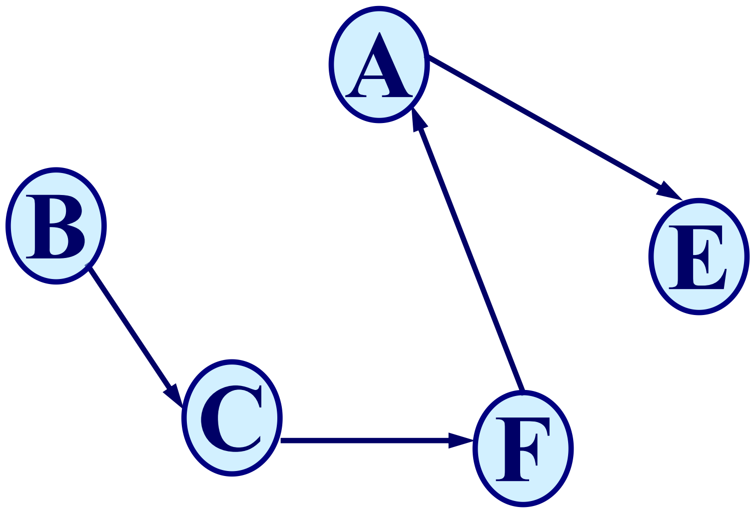
生 成 树、 生 成 森 林 ➡



弧或边带权的图分别称作**有向网**或**无向网**。

设图 $G=(V,\{VR\})$ 和图 $G'=(V',\{VR'\})$，且 $V' \subseteq V$, $VR' \subseteq VR$，则称 G' 为 G 的**子图**。





假设图中有 **n** 个顶点，**e** 条边，则

含有 **$e=n(n-1)/2$** 条边的无向图称作**完全图**；

含有 **$e=n(n-1)$** 条弧的有向图称作**有向完全图**；

若边或弧的个数 **$e<nlogn$**，则称作**稀疏图**，否则称作**稠密图**。

假若顶点v 和顶点w 之间存在一条边，则称顶点v 和w 互为**邻接点**，

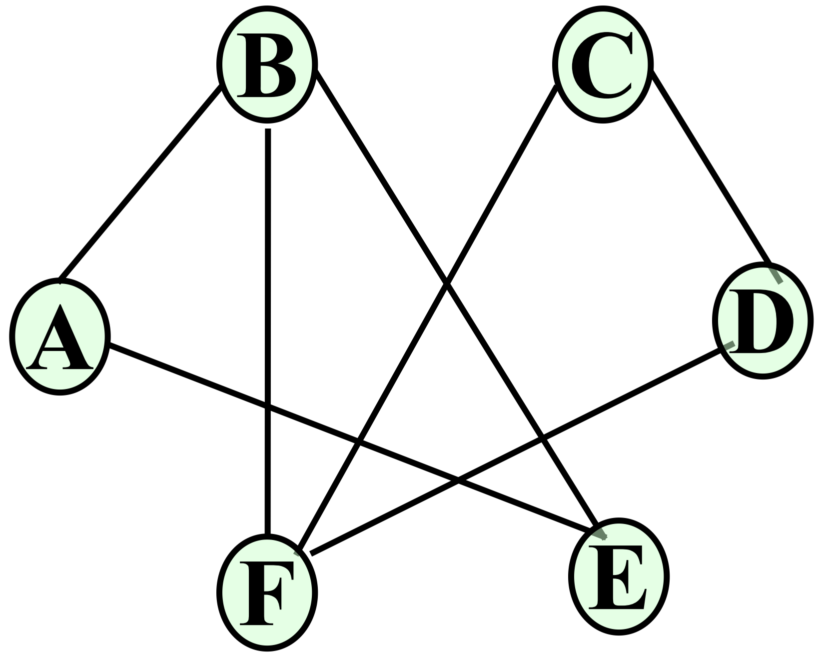
边(v,w) 和顶点v 和w 相**关联**。

和顶点v 关联的**边的数目** 定义为v的 **度**。

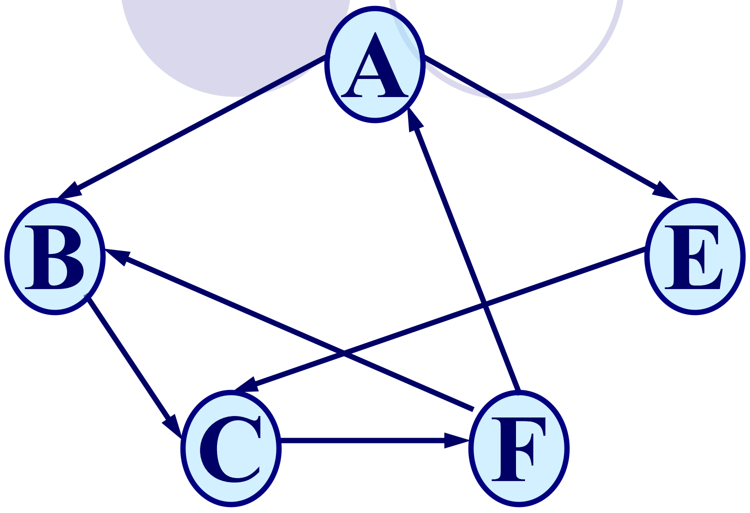
例如:

ID(B) = 3

ID(A) = 2



对有向图来说,



例如:

OD(B) = 1

ID(B) = 2

TD(B) = 3

顶点的**出 度**: 以顶点v为弧尾的弧的数目;

顶点的**入度**: 以顶点v为弧头的弧的数目。

顶点的**度(TD)=出 度(OD)+入度(ID)**

设图 G=(V,{VR})中的一个顶点序列

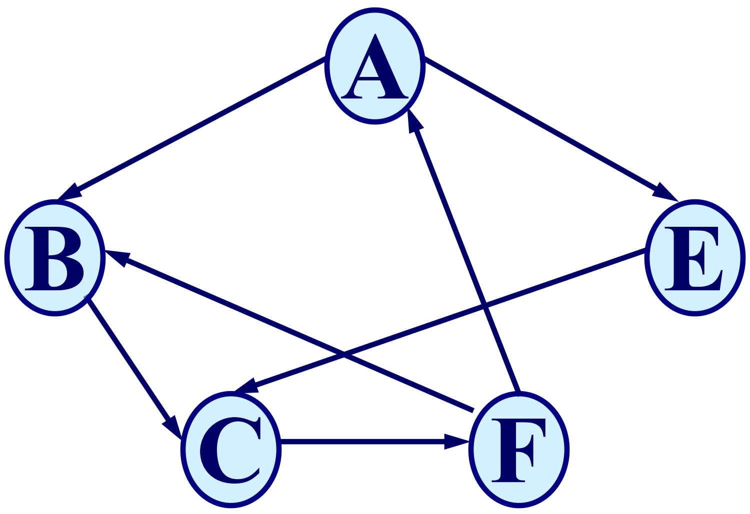
{ $u=v_{i,0},v_{i,1},\ldots,v_{i,m}=w$ }中，$(v_{i,j-1},v_{i,j})\in VR$ $1\le j\le m$,

则称从顶点u到顶点w之间存在一条**路径**。

路径上边的数目 称作**路径长度**。

如: 长度为3的路径

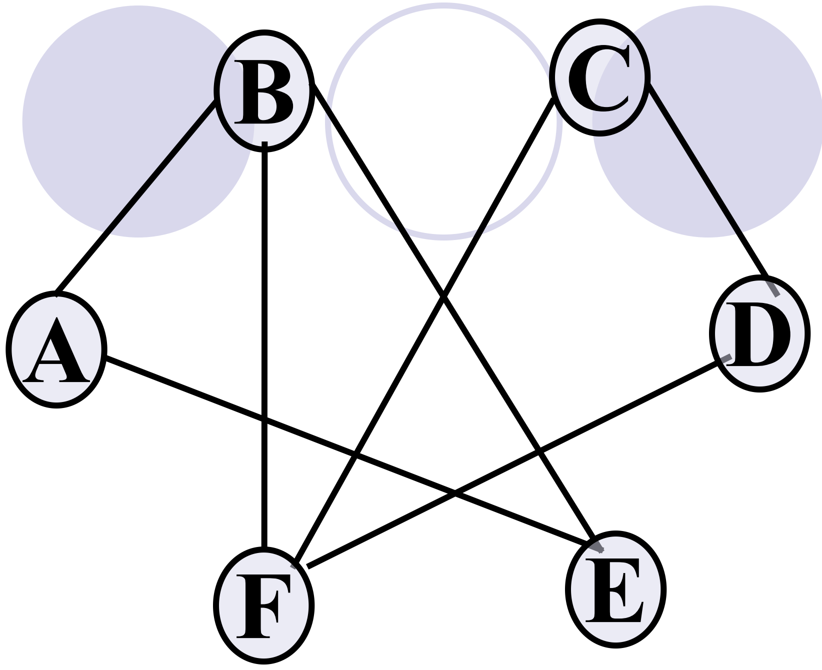
{A,B,C,F}

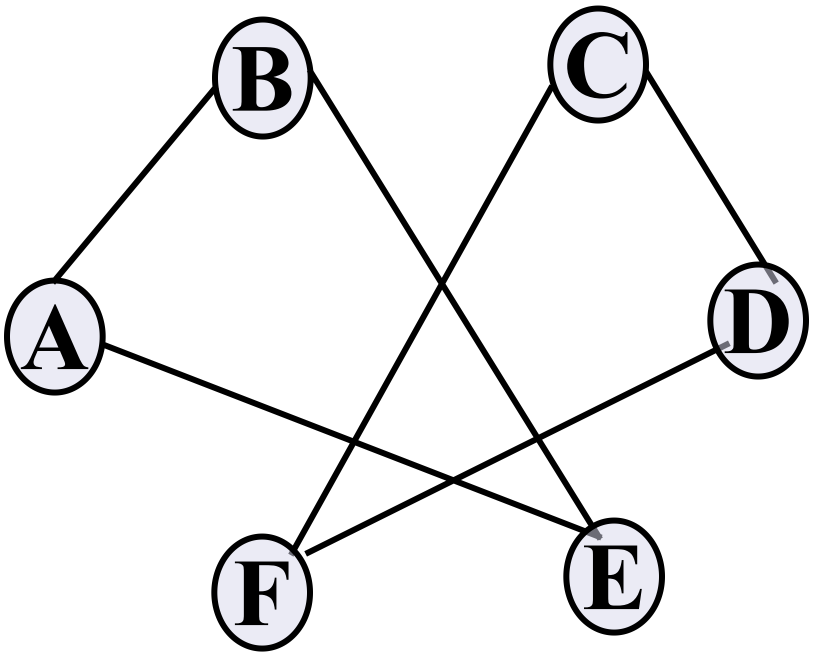


简单路径: 序列中顶点不重复出现的路径。

简单回路: 序列中第一个顶点和最后一个顶点相同的简单路径。

若图G中任意两个顶点之间都有路径相通，则称此图为**连通图**；

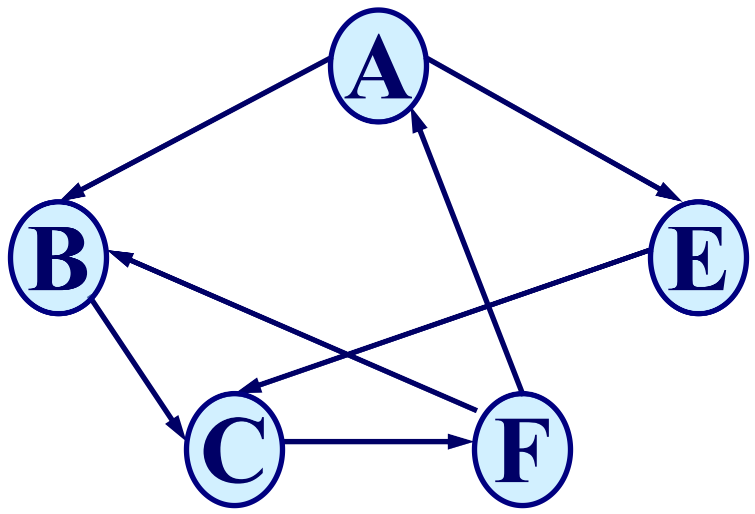


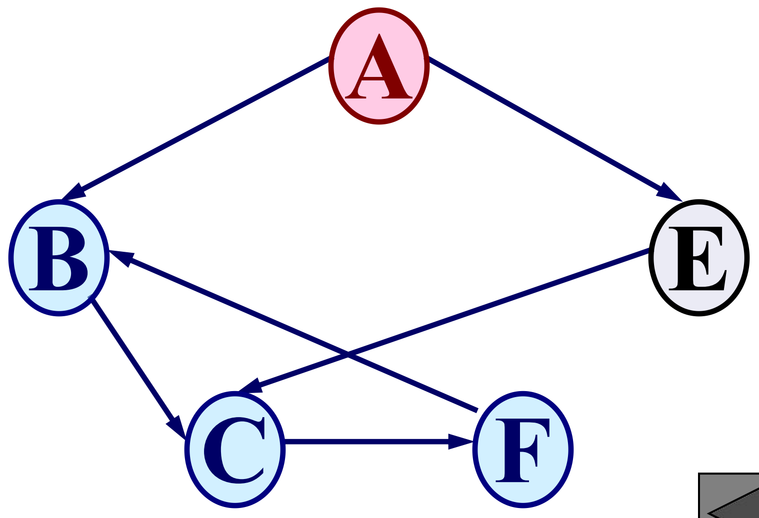


若无向图为非连通图，则图中各个极大连通子图称作此图的**连通分量**。

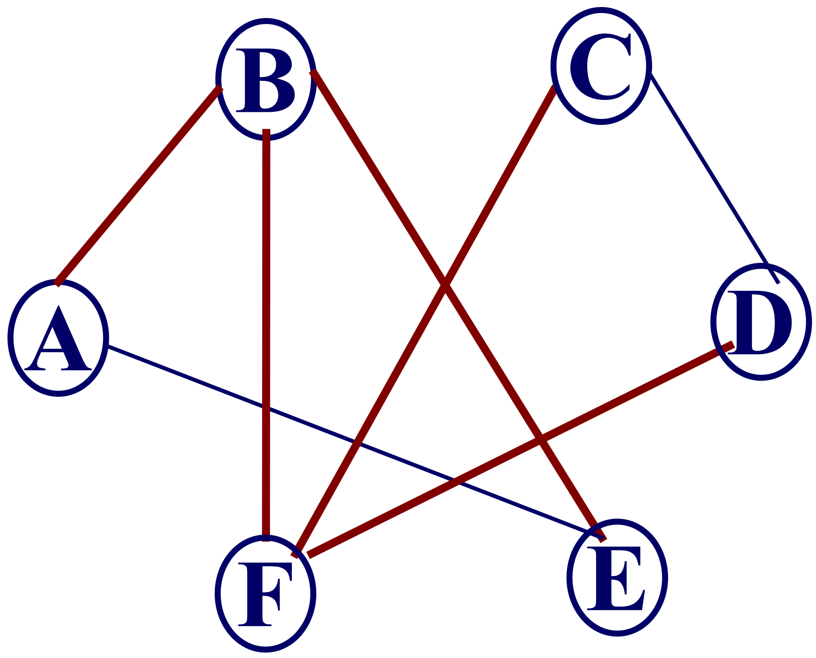
对有向图，若任意两个顶点之间都存在一条有向路径，则称此有向图为**强连通图**。

否则，其各个强连通子图称作它的**强连通分量**。





假设一个连通图有 n 个顶点和 e 条边，其中 n-1 条边和 n 个顶点构成一个极小连通子图，称该极小连通子图为此连通图的**生成树**。



对非连通图，则称由各个连通分量的生成树的集合为此非连通图的**生成森林**。

7.2 图的存储表示

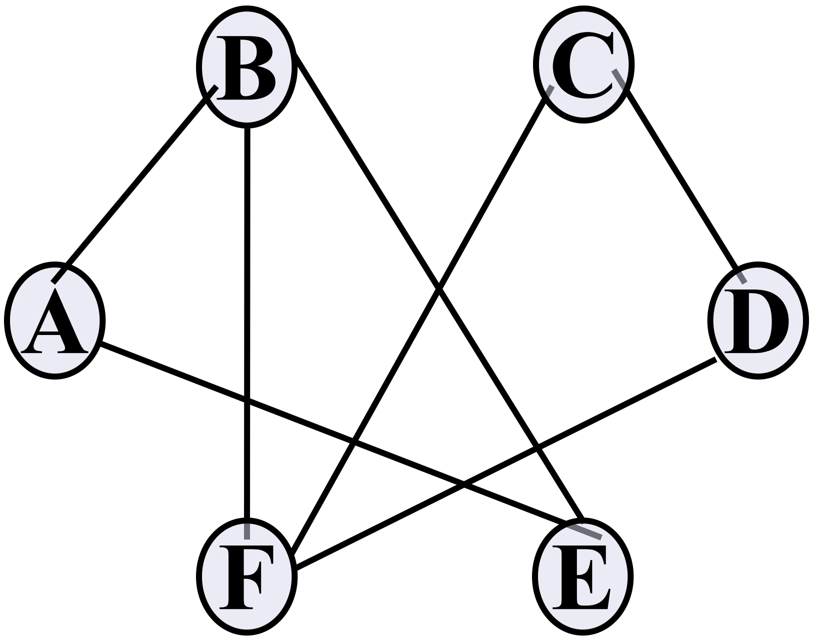
一、图的数组(邻接矩阵)存储表示

二、图的邻接表存储表示

一、图的数组(邻接矩阵)存储表示

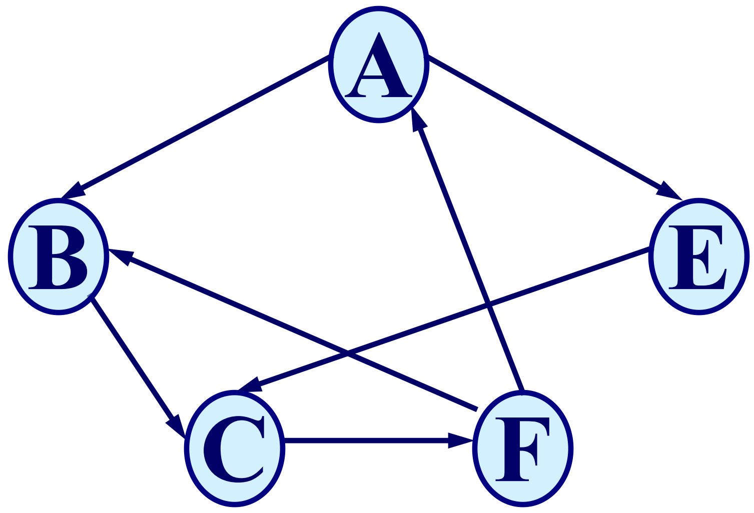
定义:矩阵的元素为

$$A_{ij}=\begin{cases}0 & (i,j)\notin VR \\ 1 & (i,j)\in VR\end{cases}$$



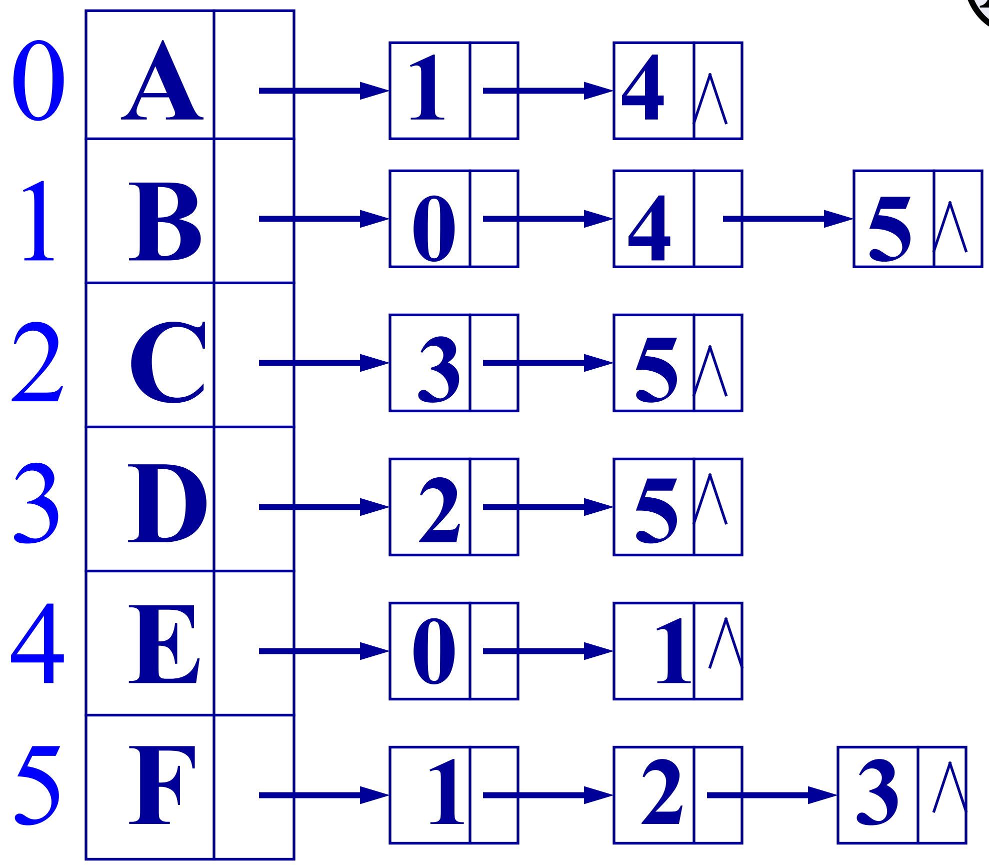
0	1	0	0	1	0
1	0	0	0	1	0
0	0	0	1	0	1
0	0	1	0	0	1
1	1	0	0	0	0
0	1	1	1	0	0

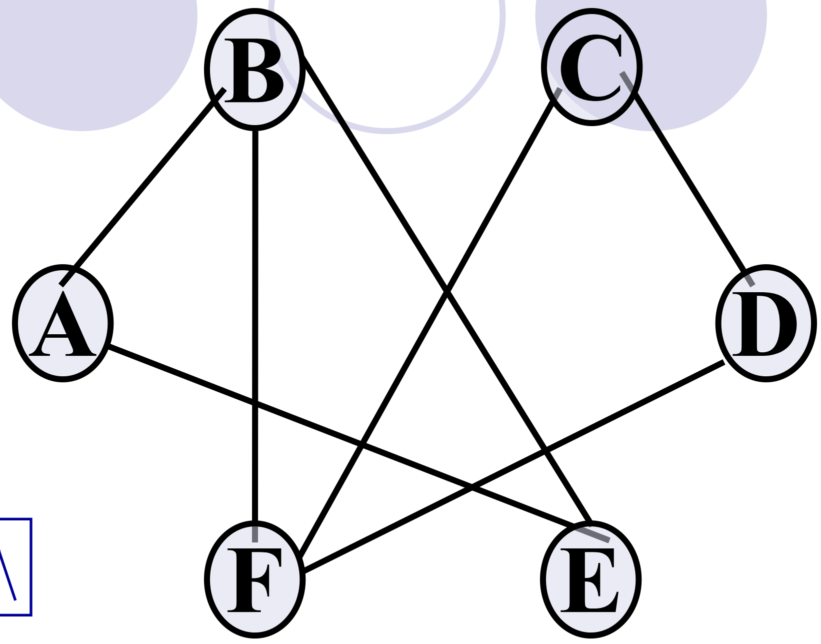
有向图的邻接矩阵为非对称矩阵



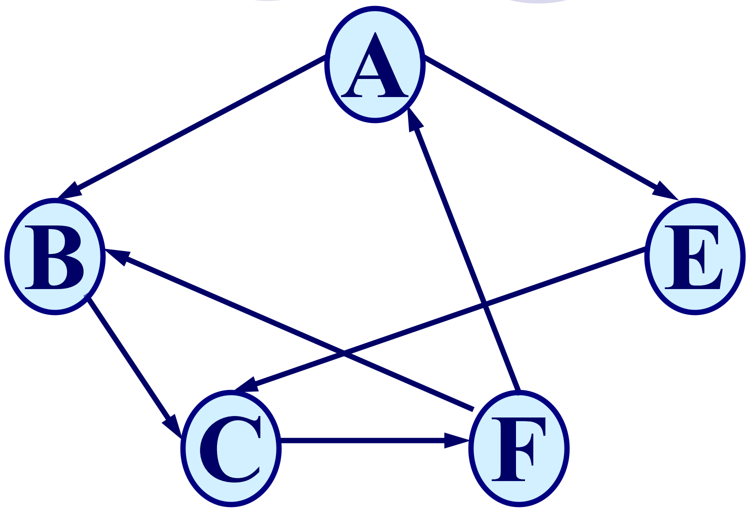
0	1	0	0	1
0	0	1	0	0
0	0	0	1	0
1	1	0	0	0
0	0	1	0	0

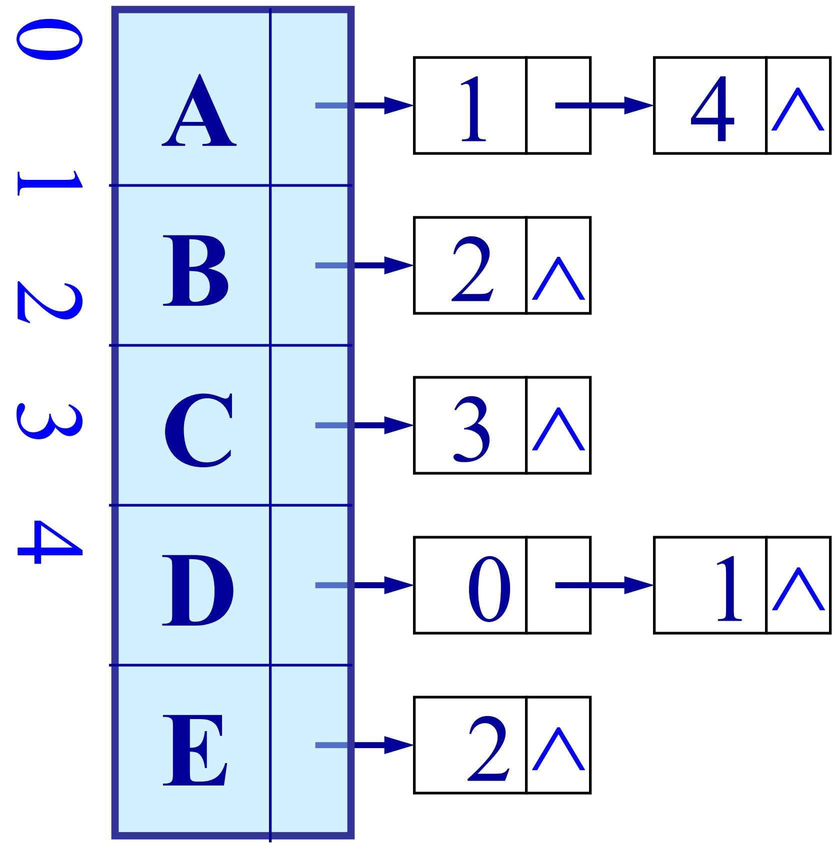
二、图的邻接表存储表示





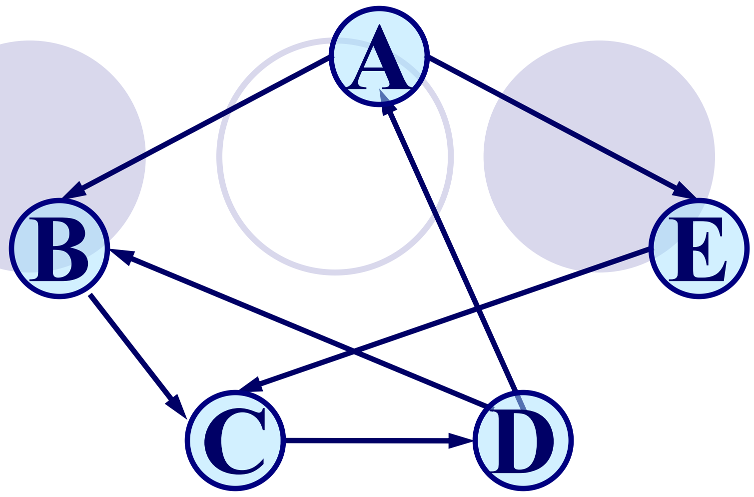
有向图的邻接表



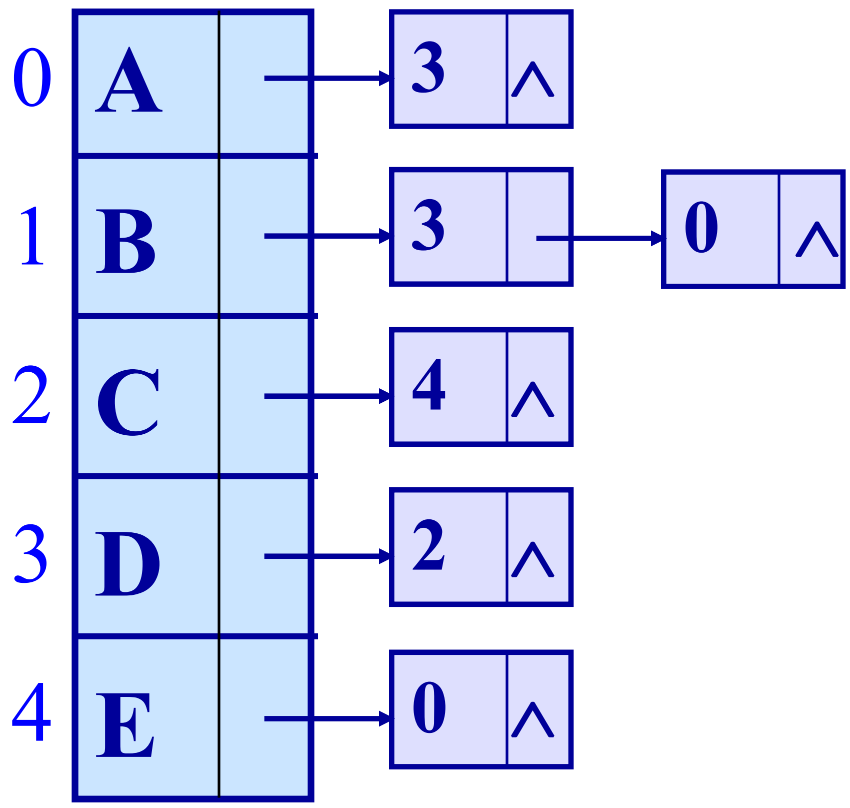


可见，在有向图的邻接表中不易找到指向该顶点的弧。

有向图的逆邻接表



在有向图的邻接表中，对每个顶点，链接的是指向该顶点的弧。



7.3图的遍历

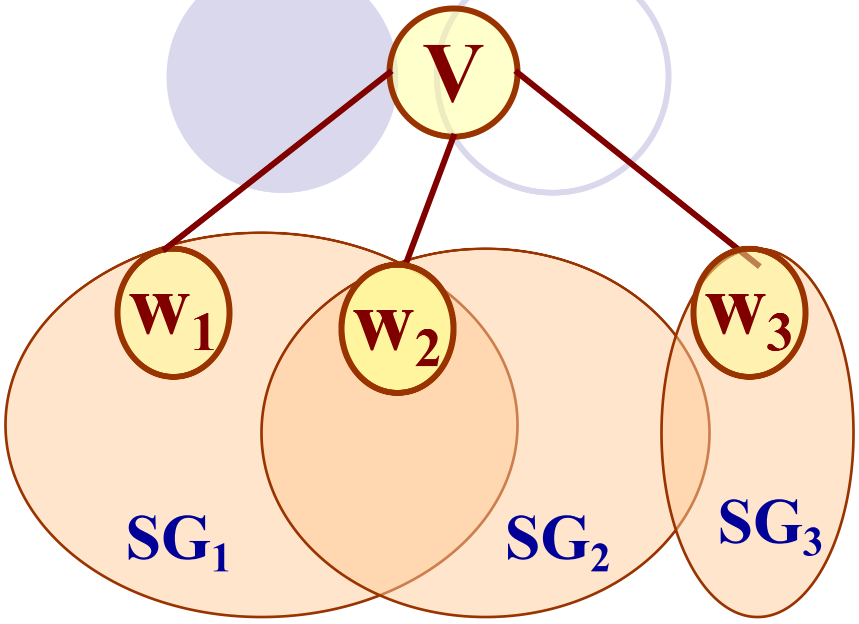
从图中某个顶点出发游历图，访遍图中其余顶点，并且使图中的每个顶点仅被访问一次的过程。

- 深度优先搜索
- 广度优先搜索
- 遍历应用举例

一、深度优先搜索遍历图

连通图的深度优先搜索遍历

从图中某个顶点V_0出发，访问此顶点，然后**依次从V_0的各个未被访问的邻接点出发深度优先搜索遍历图**，直至图中所有和V_0有路径相通的顶点都被访问到。



W_1、W_2和W_3均为V的邻接点，SG_1、SG_2和SG_3分别为含顶点W_1、W_2和W_3的子图。

访问顶点 V :

for (W_1、W_2、W_3)

若该邻接点W未被访问，

则从它出发进行深度优先搜索遍历。

从上页的图解可见:

1. 从深度优先搜索遍历连通图的过程类似于树的先根遍历;

2. 如何判别V的邻接点是否被访问?

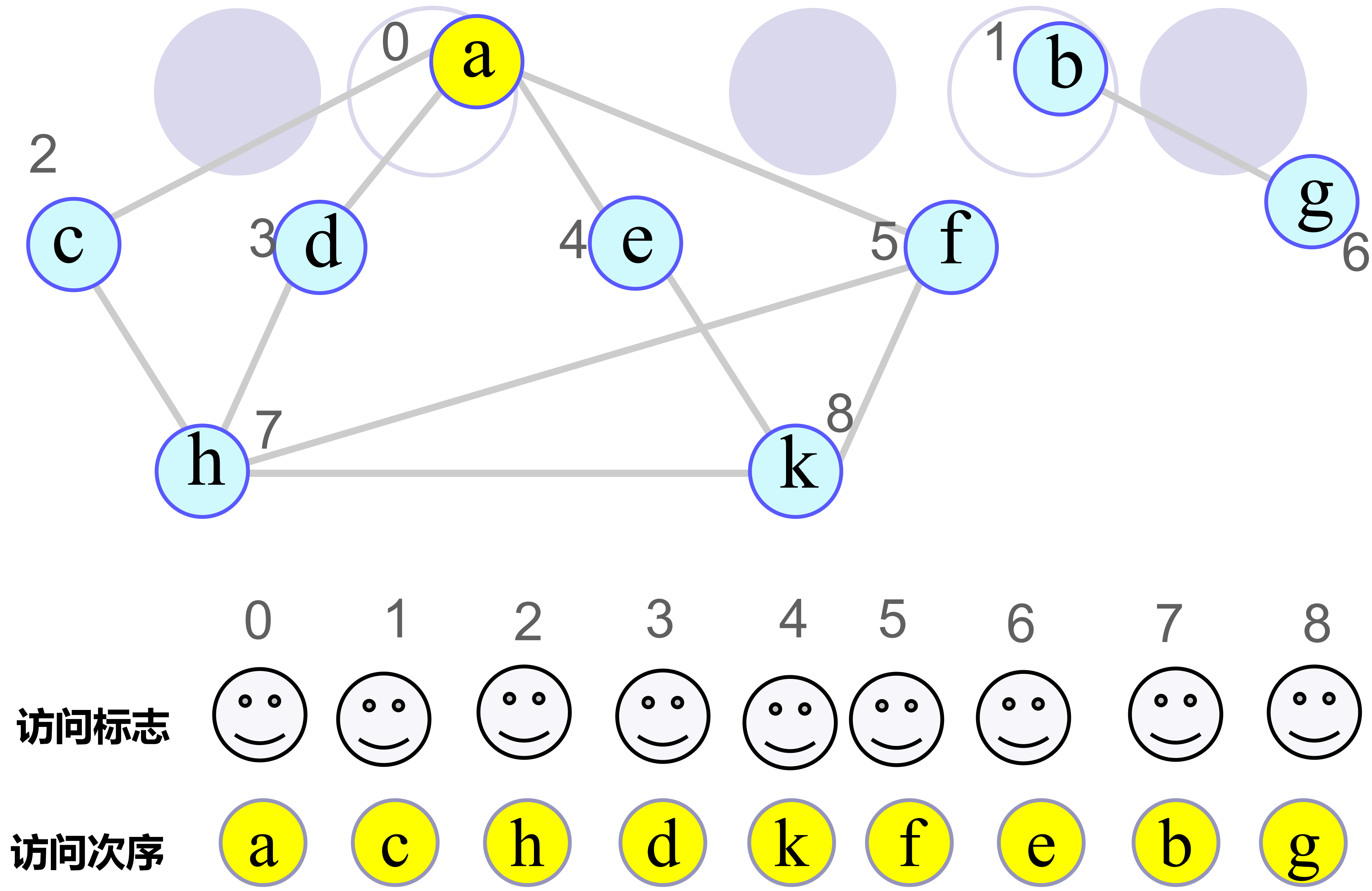
解决的办法是：为每个顶点设立一个“访问标志 visited[w]”。

```
void DFS(Graph G, int v) {
  // 从顶点v出 发，深度优先搜索遍历连通图 G
    visited[v] = TRUE;   VisitFunc(v);
    for(w=FirstAdjVex(G, v);
             w!=0; w=NextAdjVex(G,v,w))
        if (!visited[w])  DFS(G, w);
             // 对v的尚未访问的邻接顶点w
             // 递归调用DFS
} // DFS
```

非连通图的深度优先搜索遍历

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

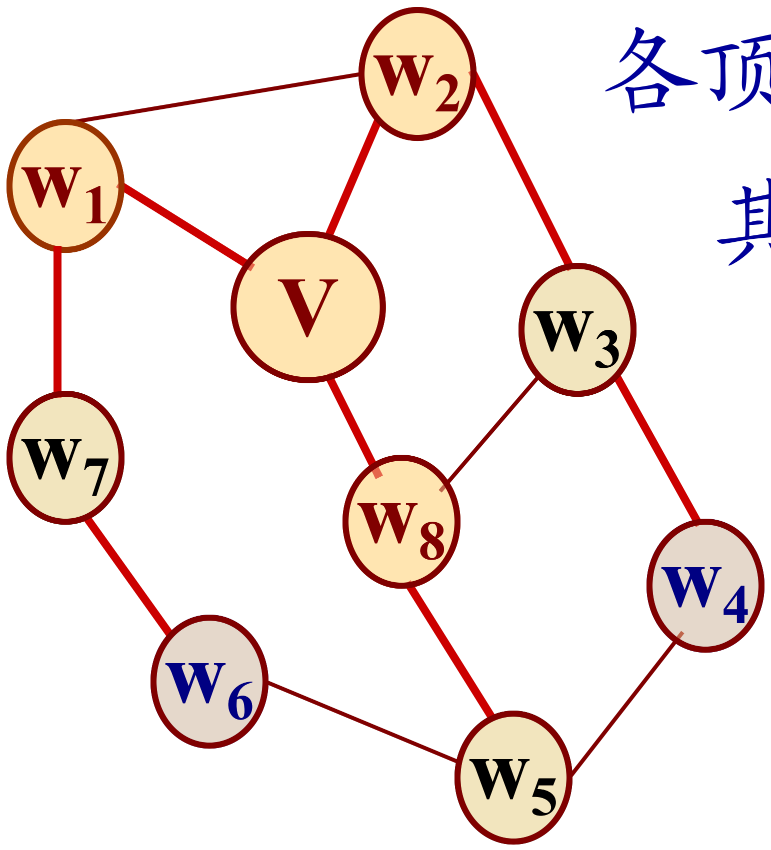
```
void DFSTraverse(Graph G,
                     Status (*Visit)(int v)) {
  // 对图 G 作深度优先遍历。
  VisitFunc = Visit;
  for (v=0; v<G.vexnum; ++v)
    visited[v] = FALSE; // 访问标志数组初始化
  for (v=0; v<G.vexnum; ++v)
    if (!visited[v])  DFS(G, v);
          // 对尚未访问的顶点调用DFS
}
```



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

二、广度优先搜索遍历图

对连通图，从起始点V到其余各顶点必定存在路径。



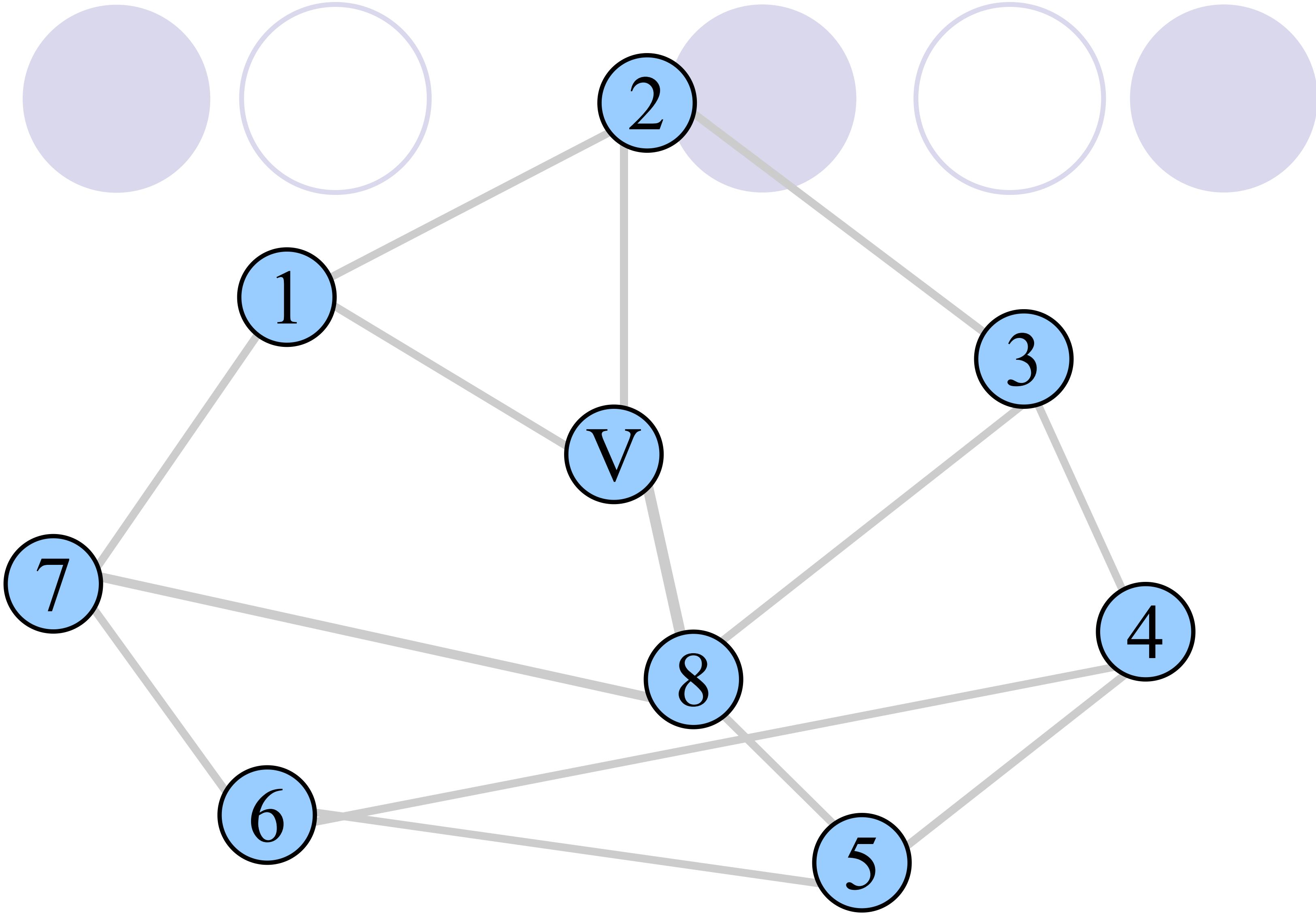
其中，V->w_1, V->w_2, V->w_8 的路径长度为1;

V->w_7, V->w_3, V->w_5 的路径长度为2;

V->w_6, V->w_4 的路径长度为3。

从图中的某个顶点V_0出发，并在访问此顶点之后依次访问V_0的所有**未被访问**过的邻接点，之后**按这些顶点被访问的先后次序依次访问它们的邻接点**，直至图中所有和V_0有路径相通的顶点都被访问到。

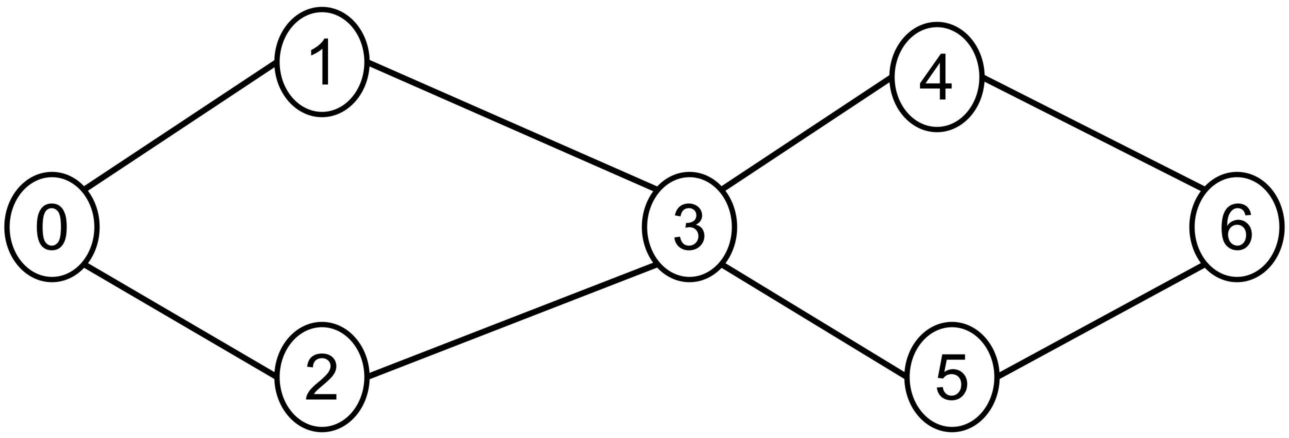
若此时图中尚有顶点未被访问，则另选图中一个未曾被访问的顶点作起始点，重复上述过程，直至图中所有顶点都被访问到为止。


2
1
3
V
7
4
8
6
5

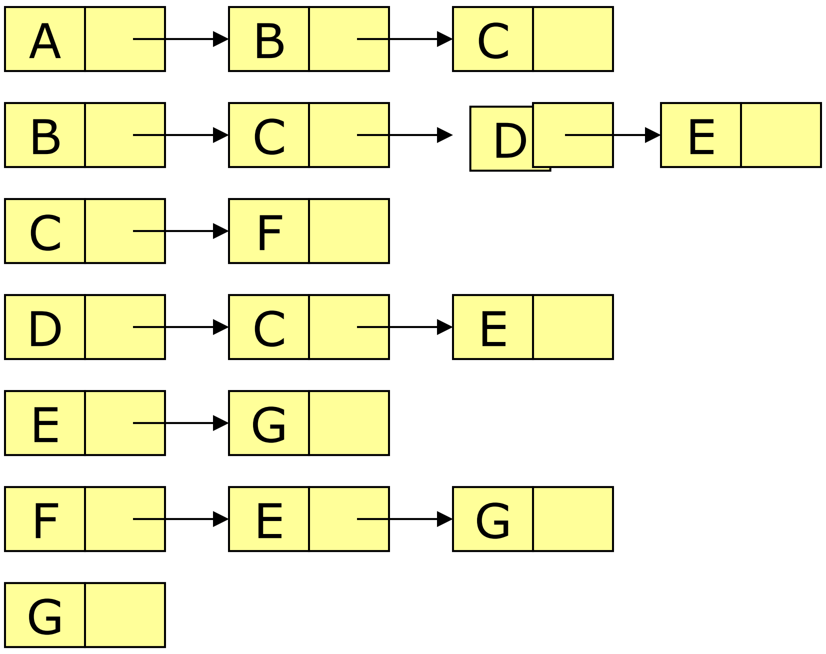
题1. 对于下图，从顶点0开始进行深度优先遍历，可得到顶点访问序列_A___。

A.0 1 3 2 4 6 5　B. 0 1 3 2 4 5 6

C.0 1 3 4 5 2 6　D. 0 1 2 3 4 6 5



- 题2. 在下面的题中，假设有某有向图的单邻接表如下：



试问以A作为开始点时：
（1）该有向图的递归算法深度优先搜索顺序是什么？f

A、ABDCFEG　　B、ACBFEDG
C、ACFGEBD　　D、ABCDEFG
E、ABDEGCF　　F、ABCFEGD

7.4 (连通网的)最小生成树

问题:

假设要在 n 个城市之间建立通讯联络网，则连通 n 个城市只需要修建 $n-1$ 条线路，**如何在最节省经费的前提下建立这个通讯网？**

该问题等价于:

构造网的一棵最小生成树，即:

在 e 条带权的边中选取 n-1 条边(不构成回路)，使“**权值之和**”为最小。

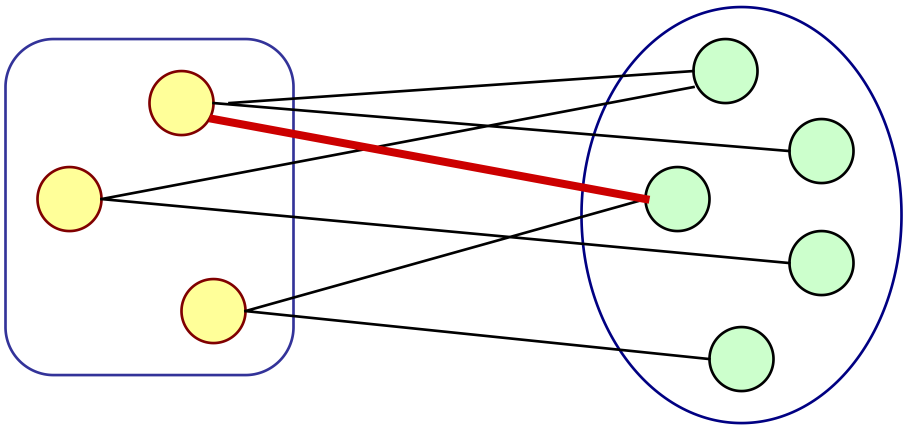
算法一:（普里姆算法）

算法二:（克鲁斯卡尔算法）

普里姆算法的基本思想：

取图中任意一个顶点 v 作为生成树的根，之后往生成树上添加新的顶点 w。**在添加的顶点 w 和已经在生成树上的顶点v 之间必定存在一条边，并且该边的权值在所有连通顶点 v 和 w 之间的边中取值最小。**之后继续往生成树上添加顶点，直至生成树上含有 n-1 个顶点为止。

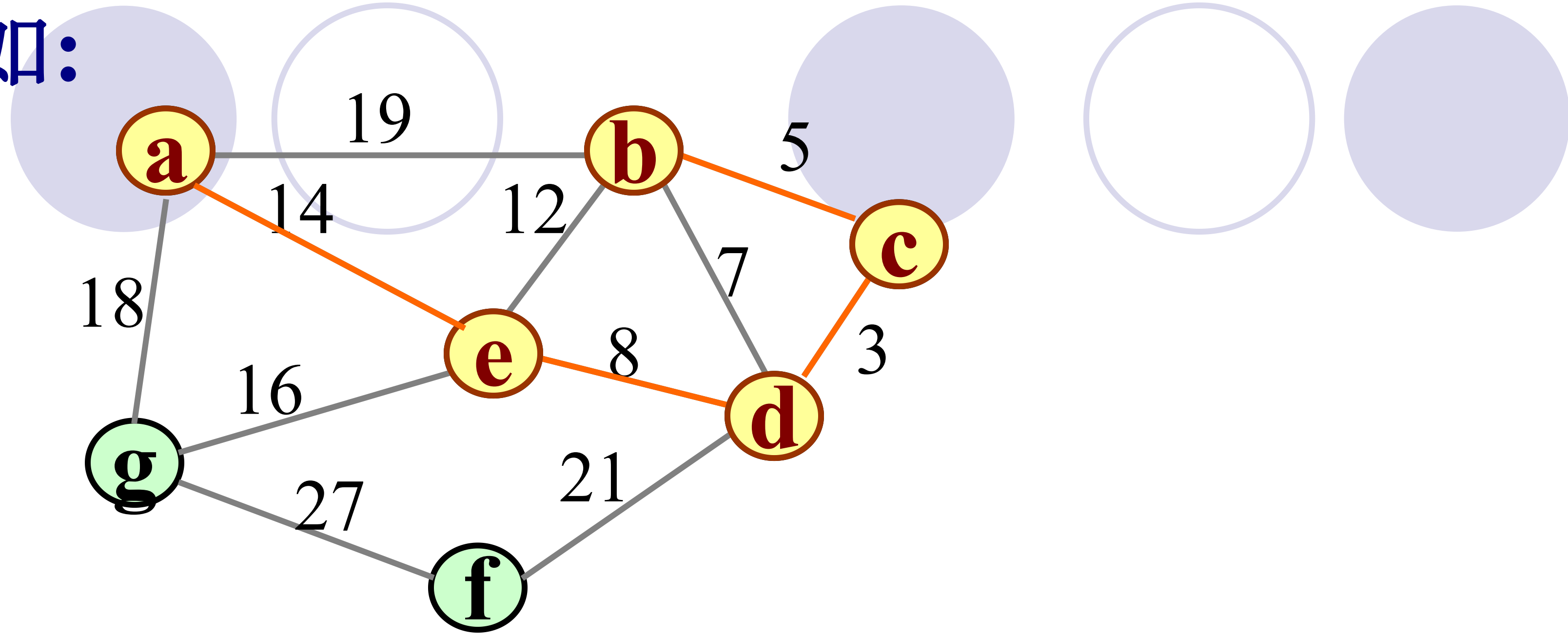
一般情况下所添加的顶点应满足下列条件：在生成树的构造过程中，图中 n 个顶点分属两个集合：已落在生成树上的顶点集 U 和尚未落在生成树上的顶点集 V-U，则应**在所有连通U中顶点和V-U中顶点的边中选取权值最小的边**。



设置一个辅助数组，对当前V- U集中的每个顶点，记录和顶点集U中顶点相连接的代价最小的边:

```
struct {
    VertexType  adjvex; // U集中的顶点序号
    VRType      lowcost; // 边的权值
} closedge[MAX_VERTEX_NUM];
```

例如:



closedge	a	b	c	d	e	f	g
Adjvex		c	d	e	a	d	e
Lowcost		**5**	**3**	**8**	**14**	21	16

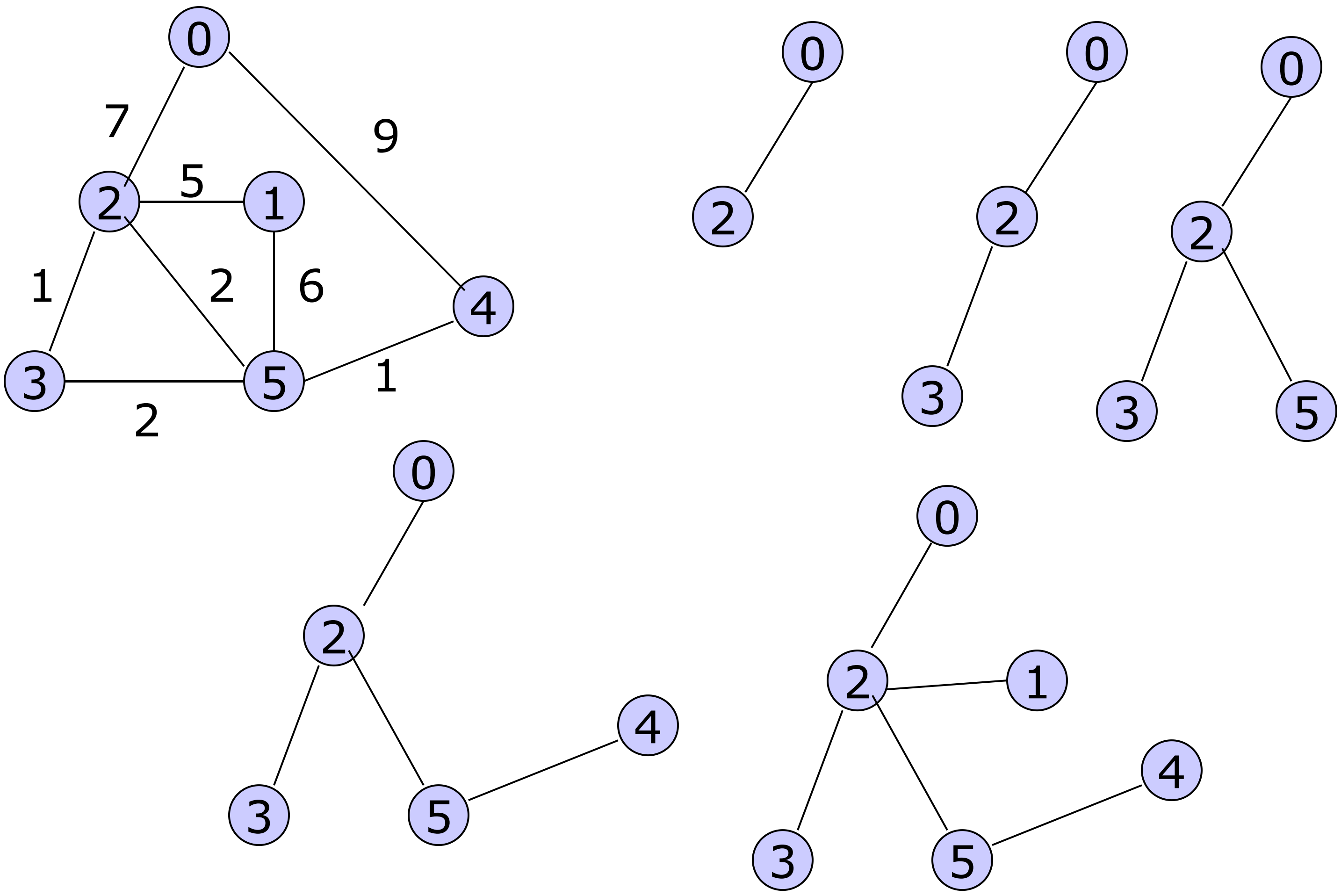
```
void MiniSpanTree_P(MGraph G, VertexType u) {
 //用普里姆算法从顶点u出发构造网G的最小生成树
  k = LocateVex ( G, u );
  for ( j=0; j<G.vexnum; ++j )  // 辅助数组初始化
     if (j!=k)
        closedge[j] = { u, G.arcs[k][j].adj };
  closedge[k].lowcost = 0;      // 初始， U= {u}
  for (i=0; i<G.vexnum; ++i) {
     继续向生成树上添加顶点;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

```
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].adj < closedge[j].lowcost)
    closedge[j] = { G.vexs[k], G.arcs[k][j].adj };
```

- 题3：
- 若一个带权无向图的邻接矩阵如下表示：

$$
\begin{bmatrix}
0 & \infty & 7 & \infty & 9 & \infty \\
\infty & 0 & 5 & \infty & \infty & 6 \\
7 & 5 & 0 & 1 & \infty & 2 \\
\infty & \infty & 1 & 0 & \infty & 2 \\
9 & \infty & \infty & \infty & 0 & 1 \\
\infty & 6 & 2 & 2 & 1 & 0
\end{bmatrix}
$$

- 画出该树，并按prim算法构造该图的一棵最小生成树。


0
7
9
2
5
1
1
2
6
4
3
5
1
2

克鲁斯卡尔算法的基本思想:

- **考虑问题的出发点:** 为使生成树上边的权值之和达到最小，则应使生成树中每一条边的权值尽可能地小。

- **具体做法:** 先构造一个只含 n 个顶点的子图 SG，然后从权值最小的边开始，若它的添加不使SG中产生回路，则在 SG 上加上这条边，如此重复，直至加上 n-1 条边为止。

算法描述:

构造非连通图 ST=(V,{ });

k = i = 0;   // k 计选中的边数

while (k<n-1) {

++i;

检查边集 E 中**第 i 条权值最小的边**(u,v);

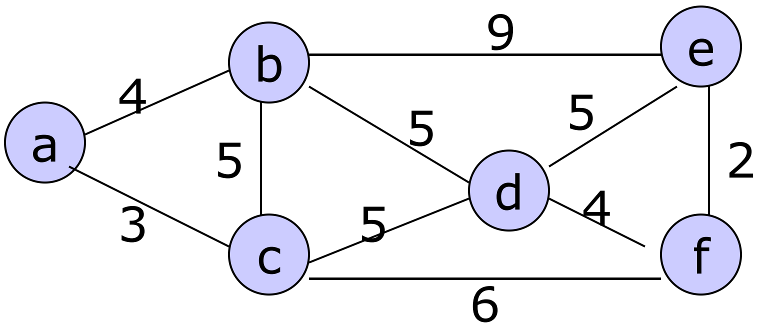
若(u,v)加入ST后不使ST中 产生回路,

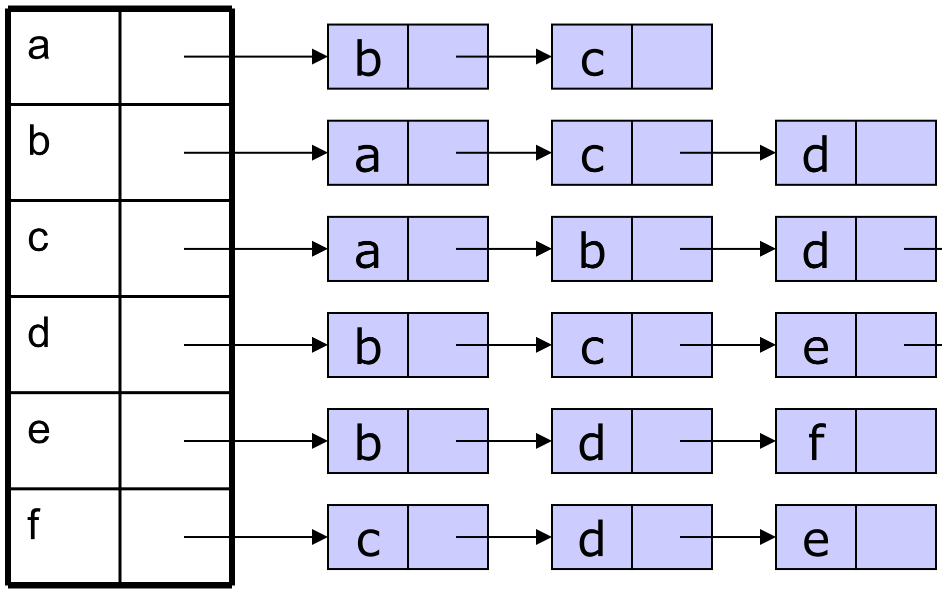
则 输出 边(u,v); 且 k++;

}

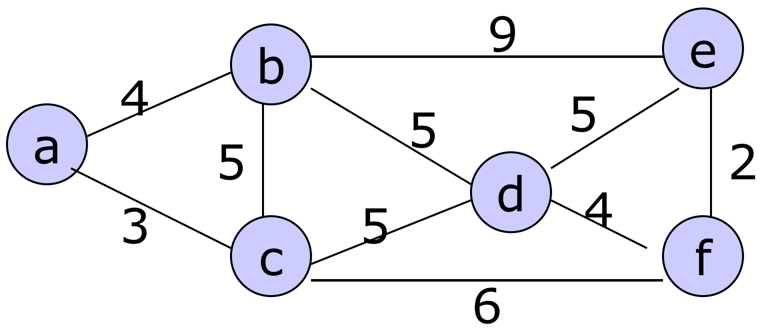
例如:

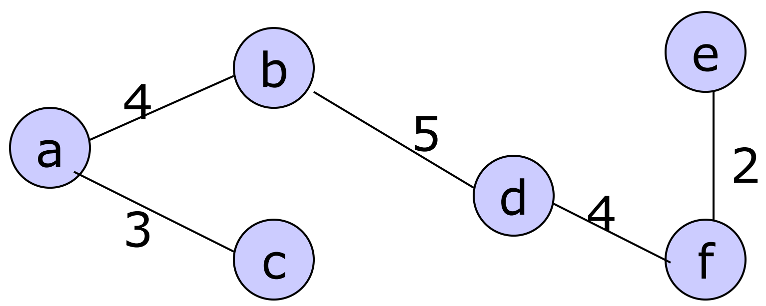
- 题4：
- 对右图所示的无向加权图完成下列要求：
- （1）写出它的邻接表和邻接矩阵；
- （2）按Kruskal算法求出它的最小生成树并演示其过程；
- （3）求其最小生成树的代价WG（T）。





$$\begin{bmatrix} 0 & 4 & 3 & \infty & \infty & \infty \\ 4 & 0 & 5 & 5 & 9 & \infty \\ 3 & 5 & 0 & 5 & \infty & 6 \\ \infty & 5 & 5 & 0 & 5 & 4 \\ \infty & 9 & \infty & 5 & 0 & 2 \\ \infty & \infty & 6 & 4 & 2 & 0 \end{bmatrix}$$





WT(G)=2+4+5+3+4=18

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

7.5.1 拓扑排序

问题:

假设以有向图表示一个工程的施工图或程序的数据流图，则图中不允许出现回路。

检查有向图中是否存在回路的方法之一，是对有向图进行**拓扑排序**。

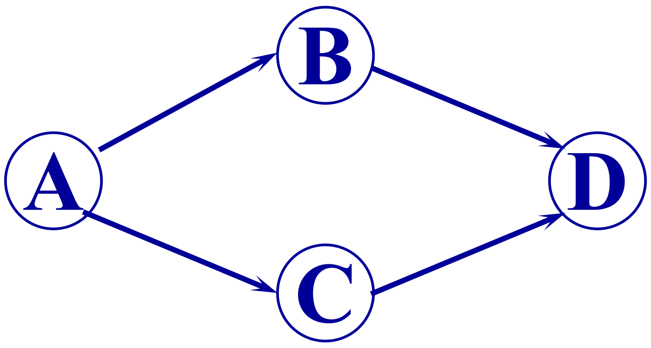
何谓“拓扑排序”？

对有向图进行如下操作:

按照有向图给出的次序关系，将图中顶点排成一个线性序列，对于有向图中没有限定次序关系的顶点，则可以人为加上任意的次序关系。

由此所得顶点的线性序列称之为**拓扑有序序列**

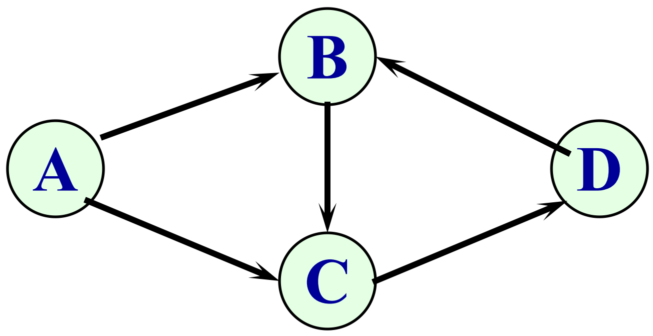
例如：对于下列有向图



可求得拓扑有序序列：

A B C D 或 A C B D

反之，对于下列有向图



不能求得它的拓扑有序序列。

因为图中存在一个回路 {B, C, D}

题9：若一个有向图中的顶点不能排成一个拓扑序列，则可断定该有向图____。D

A.是一个有根的有向图　　　　B. 是一个强连通图

C. 含有多个入度为0的顶点

D. 含有顶点数目大于1的强连通分量（…局部有环）

如何进行拓扑排序?

一、从有向图中选取一个**没有前驱**的顶点，并输出之;

二、从有向图中**删去此顶点以及所有以它为尾的弧**;

重复上述两步，直至图空，或者图不空但找不到无前驱的顶点为止。

a　b　h　c　d　g　f　e

在算法中需要用定量的描述替代定性的概念

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

算法描述

```
取入度为零的顶点v;
while (v<>0) {
    printf(v);  ++m;
    w:=FirstAdj(v);
    while (w<>0) {
      inDegree[w]--;
      w:=nextAdj(v,w);
    }
    取下一个入度为零的顶点v;
}
if m<n  printf("图中有回路");
```

为避免每次都要搜索入度为零的顶点，在算法中设置一个“**栈**”，以保存“入度为零”的顶点。

```
CountInDegree(G,indegree);
      //对各顶点求入度
InitStack(S);
for ( i=0; i<G.vexnum; ++i)
   if (!indegree[i])  Push(S, i);
      //入度为零的顶点入栈
```

```
count=0;          //对输出顶点计数
while (!EmptyStack(S)) {
  Pop(S, v);  ++count;  printf(v);
  for (w=FirstAdj(v); w;  w=NextAdj(G,v,w)){
     --indegree(w);  // 弧头顶点的入度减一
     if (!indegree[w])  Push(S, w);
            //新产生的入度为零的顶点入栈
  }
}
if (count<G.vexnum) printf("图中有回路")
```

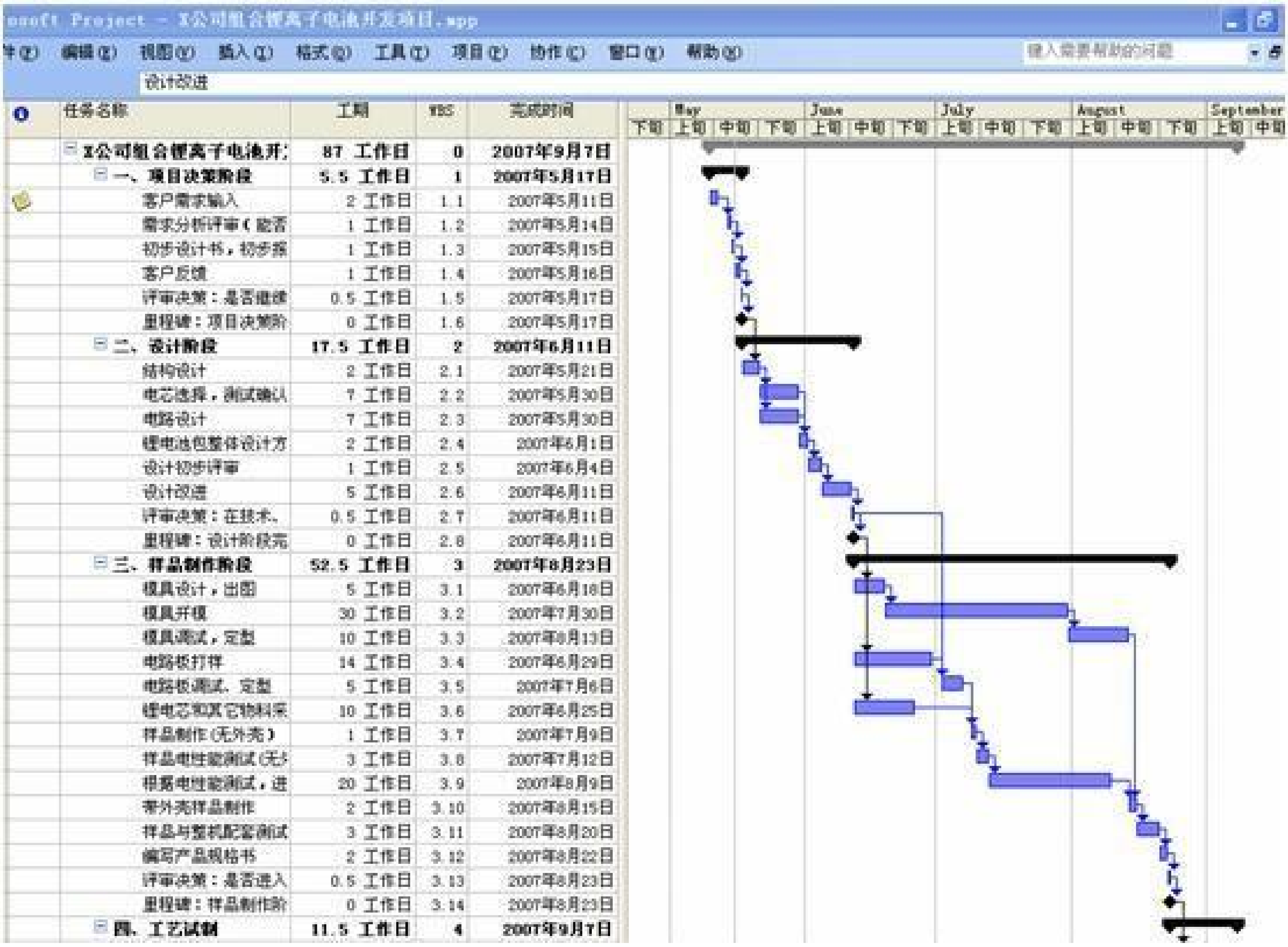
7.5.2 关键路径

问题：

假设以有向网表示一个施工流图，弧上的权值表示完成该项子工程所需时间。

问：哪些子工程项是“关键工程”？

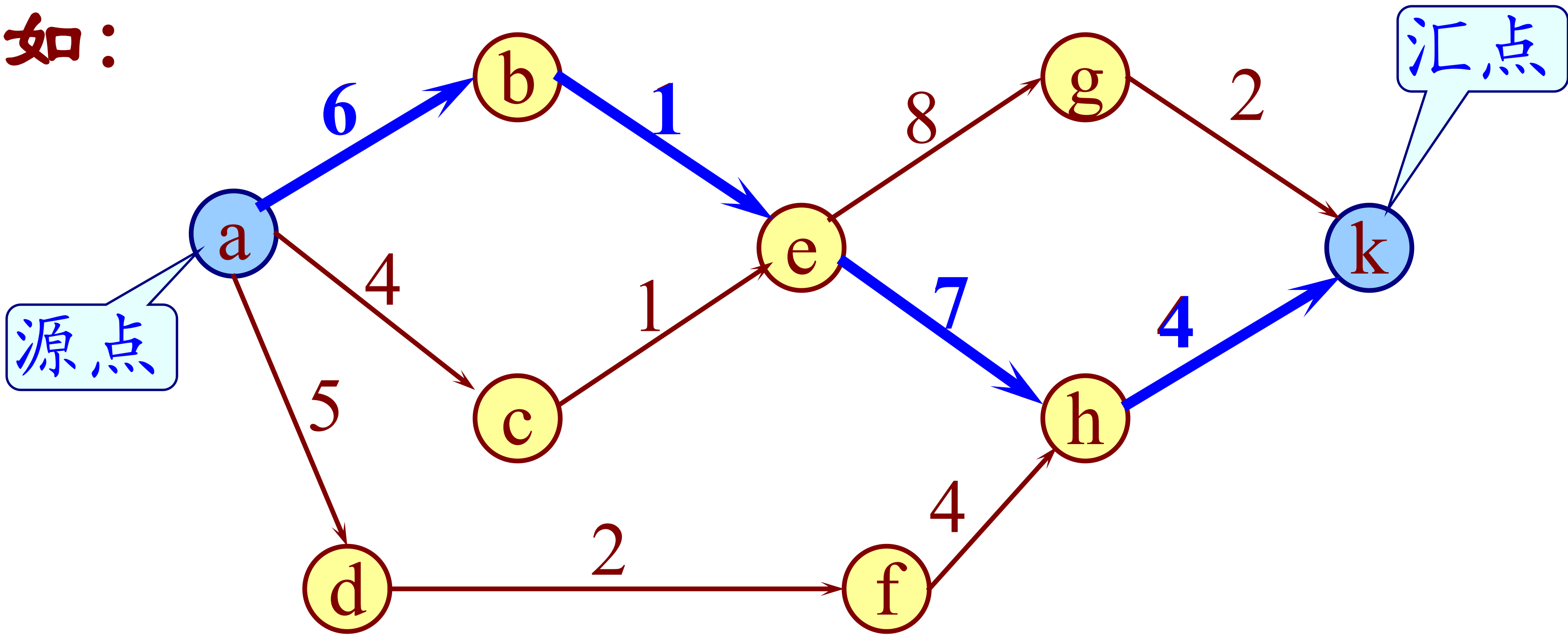
即：哪些子工程项将影响整个工程的完成期限的。



任务名称	工期	WBS	完成时间
X公司组合锂离子电池开	**87 工作日**	**0**	**2007年9月7日**
一、项目决策阶段	**5.5 工作日**	**1**	**2007年5月17日**
客户需求输入	2 工作日	1.1	2007年5月11日
需求分析评审（能否	1 工作日	1.2	2007年5月14日
初步设计书，初步报	1 工作日	1.3	2007年5月15日
客户反馈	1 工作日	1.4	2007年5月16日
评审决策：是否继续	0.5 工作日	1.5	2007年5月17日
里程碑：项目决策阶	0 工作日	1.6	2007年5月17日
二、设计阶段	**17.5 工作日**	**2**	**2007年6月11日**
结构设计	2 工作日	2.1	2007年5月21日
电芯选择，测试确认	7 工作日	2.2	2007年5月30日
电路设计	7 工作日	2.3	2007年5月30日
锂电池包整体设计方	2 工作日	2.4	2007年6月1日
设计初步评审	1 工作日	2.5	2007年6月4日
设计改进	5 工作日	2.6	2007年6月11日
评审决策：在技术、	0.5 工作日	2.7	2007年6月11日
里程碑：设计阶段完	0 工作日	2.8	2007年6月11日
三、样品制作阶段	**52.5 工作日**	**3**	**2007年8月23日**
模具设计，出图	5 工作日	3.1	2007年6月18日
模具开模	30 工作日	3.2	2007年7月30日
模具调试，定型	10 工作日	3.3	2007年8月13日
电路板打样	14 工作日	3.4	2007年6月29日
电路板调试、定型	5 工作日	3.5	2007年7月6日
锂电芯和其它物料采	10 工作日	3.6	2007年6月25日
样品制作(无外壳）	1 工作日	3.7	2007年7月9日
样品电性能测试(无タ	3 工作日	3.8	2007年7月12日
根据电性能测试，进	20 工作日	3.9	2007年8月9日
带外壳样品制作	2 工作日	3.10	2007年8月15日
样品与整机配套测试	3 工作日	3.11	2007年8月20日
编写产品规格书	2 工作日	3.12	2007年8月22日
评审决策：是否进入	0.5 工作日	3.13	2007年8月23日
里程碑：样品制作阶	0 工作日	3.14	2007年8月23日
四、工艺试制	**11.5 工作日**	**4**	**2007年9月7日**

整个工程完成的时间为：从有向图的**源点**到**汇点**的最长路径。

例如：



"关键活动"指的是：该弧上的**权值增加**将使有向图上的**最长路径的长度增加。**

如何求关键活动?

“ 事件(顶点)”的 最早发生时间 $ve(j)$

ve(j) = 从源点到顶点j的最长路径长度;

“ 事件(顶点)”的 最迟发生时间 $vl(k)$

vl(k) = 从顶点k到汇点的最短路径长度。

1.最早发生时间：从前往后，前驱结点到当前结点所需时间，取最大值MAX。
2.最迟发生时间：从后往前，后继结点的最迟发生时间-边权值，取最小值MIN。
3.关键路径：最早发生时间和最迟发生时间相同的结点即为关键路径上的节点。
4.最早开始时间：等于当前边起始结点的最早发生时间。
5.最晚开始时间：等于当前边指向结点的最迟发生时间-当前边的权值。
6.最早完工时间：等于当前边指向结点的最早发生时间。
7.最晚完工时间：等于当前边指向结点的最迟发生时间。

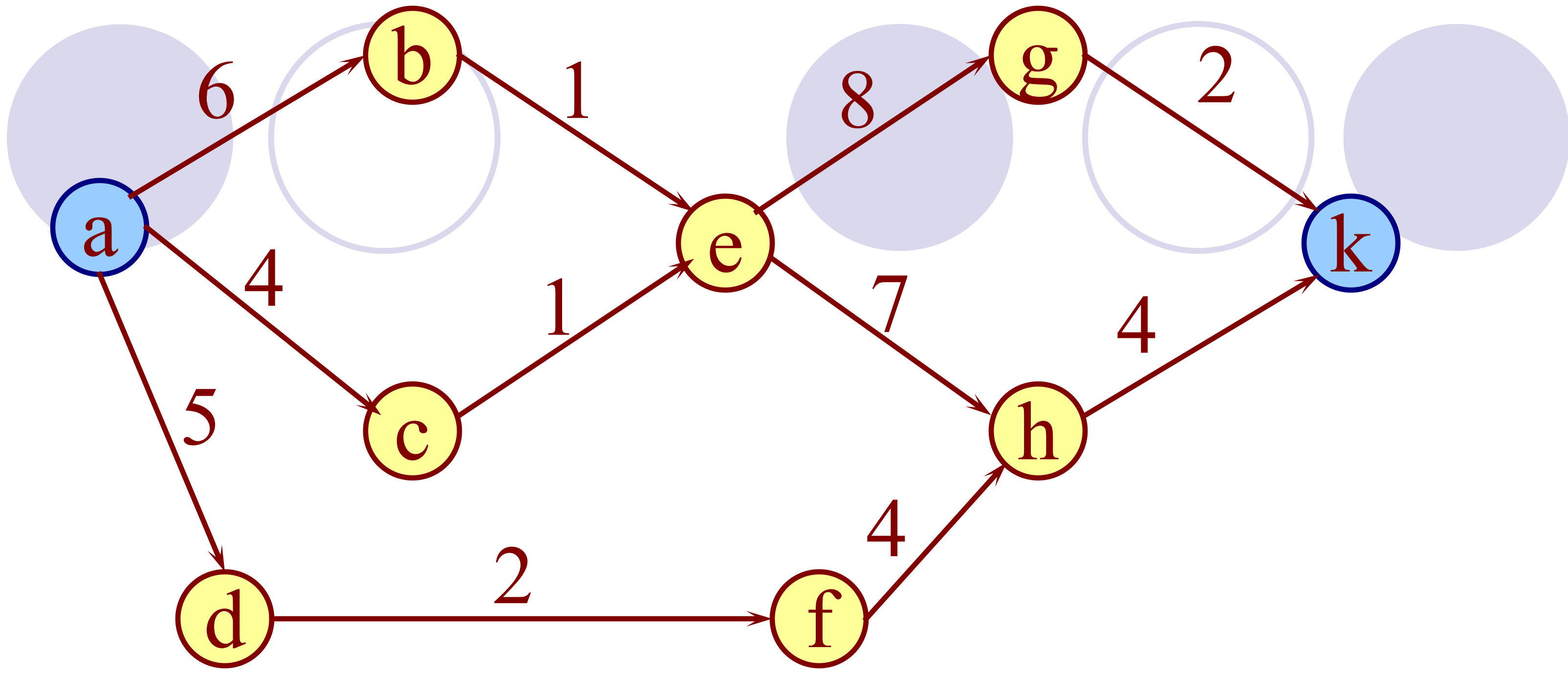
事件发生时间的计算公式:

$$ve(\text{源点}) = 0;$$

$$ve(k) = \mathrm{Max}\{ve(j) + dut(<j, k>)\}$$

$$vl(\text{汇点}) = ve(\text{汇点});$$

$$vl(j) = \mathrm{Min}\{vl(k) - dut(<j, k>)\}$$



	a	b	c	d	e	f	g	h	k
ve	0	6	4	5	7	7	15	14	18
vl	0	6	6	8	7	10	16	14	18

拓扑有序序列：a - d - f - c - b - e - h - g - k

	a	b	c	d	e	f	g	h	k
ve	0	6	4	5	7	7	15	14	18
vl	0	6	6	8	7	10	16	14	18

	ab	ac	ad	be	ce	df	eg	eh	fh	gk	hk
权	6	4	5	1	1	2	8	7	4	2	4
e	0	0	0	6	4	5	7	7	7	15	14
l	0	2	3	6	6	8	8	7	10	16	14
	✓			✓				✓			✓

算法的实现要点：

显然，求ve的顺序应该是按**拓扑有序**的次序；

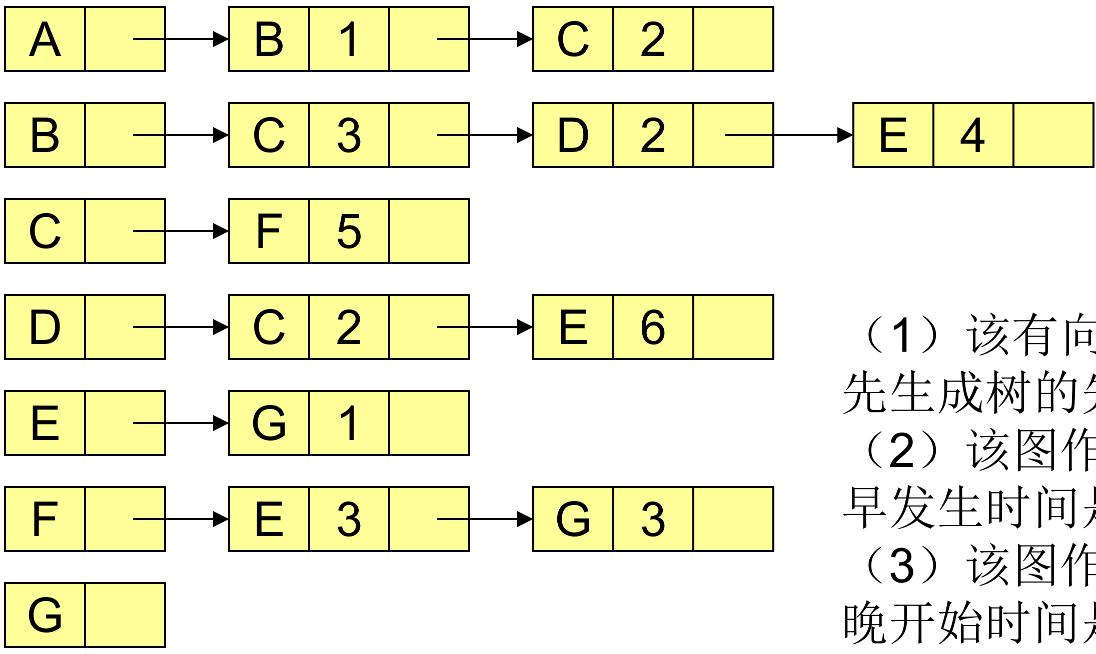
而　　求vl的顺序应该是按**拓扑逆序**的次序；

因为　拓扑逆序序列即为拓扑有序序列的**逆序列**，

因此　应该在拓扑排序的过程中，

另设一个“**栈**”记下拓扑有序序列。

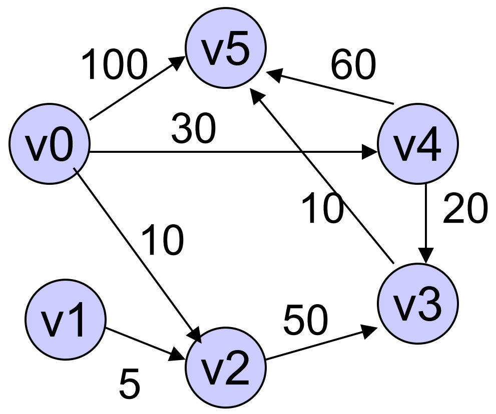
- 题10：
- 5，9？



（1）该有向图（不考虑权值）的广度优先生成树的先序遍历是什么？
（2）该图作为AOE网络图时，事件C的最早发生时间是什么？
（3）该图作为AOE网络图时，活动BE的最晚开始时间是什么？

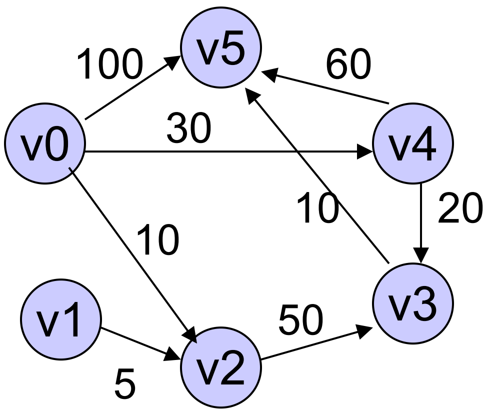
7.6 最短路径

- 7.6.1 从某个源点到其余各顶点的最短路径
- 先讨论单源点的最短路径问题：给定带权有向图G和源点v，求从v到G中其余各顶点的最短路径。



始点	终点	最短路径	路径长度
v0	v1	无	
	v2	(v0,v2)	10
	v3	(v0,v4,v3)	50
	v4	(v0,v4)	30
	v5	(v0,v4,v3,v5)	60

- 如何求得这些路径？Dijkstra提出了一个按路径长度递增的次序产生最短路径的算法。
- 首先引进一个辅助向量D，它的每个分量D[i]表示当前所找到的从结点v到每个终点vi的最短路径的长度。它的初态为：若从v到vi有弧，则D[i]为弧上的权值；否则置D[i]为∞
- 带权邻接矩阵为：



$$\begin{bmatrix} \infty & \infty & 10 & \infty & 30 & 100 \\ \infty & \infty & 5 & \infty & \infty & \infty \\ \infty & \infty & \infty & 50 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & 10 \\ \infty & \infty & \infty & 20 & \infty & 60 \\ \infty & \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$

- 显然，长度为 $D[j] = \underset{i}{Min}\{D[i] \mid v_i \in V\}$ 的路径就是从v出发的长度最短的一条路径长度。
- 那么，下一条长度次短的最短路径是哪一条呢？假设该次短路径的终点是vk，则可想而知，这条路径或者是（v，vk），或者是（v，vj，vk）。它的长度或者是从v到vk的弧上的权值，或者是D[j]和从vj到vk的弧上的权值之和。
- 一般情况下，假设S为已求得最短路径的终点的集合，则可证明：下一条最短路径（设其终点为x）或者是弧（v，x），或者是中间只经过S中的顶点而最后到达顶点x的路径。
- 因此，下一条长度次短的最短路径长度必是

$$D[j] = \underset{i}{Min}\{D[i] \mid v_i \in V - S\}$$

其中，D[i]或者是弧（v，vi）上的权值，或者是D[k]和弧（vk，vi）的权值之和。

- 根据以上分析，可以得到如下描述的算法：
- （1）假设用带权的邻接矩阵arcs来表示带权有向图，arcs[i][j]表示弧（vi，vj）上的权值，若<vi,vj>不存在，则置arcs[i][j]=∞
- S为已找到从v出发的最短路径的终点的集合，它的初始状态为空集。那么，从v出发到图上其余顶点vi可能达到的最短路径长度初值为
- D[i]=arcs[LocateVex(G,v)][i];
- (2)选择vj，使得

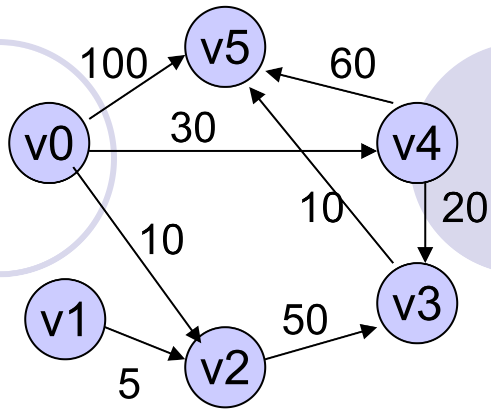
$$D[j] = \underset{i}{Min}\{D[i] \mid v_i \in V - S\}$$

- vj 就是当前求得的一条从v出发的最短路径的终点，令S= S U {j}

（3）修改从v出发到集合V-S上任一顶点vk可达的最短路径长度，如果 D[j]+arcs[j][k]<D[k]

则修改D[k]为D[k] = D[j]+arcs[j][k]

（4）重复操作（2）、（3）共n-1次。



终点	从v0到各结点的D值和最短路径的求解过程				
	i=1	i=2	i=3	i=4	i=5
v1	∞	∞	∞	∞	∞
v2	10(v0,v2)				
v3	∞	60(v0,v2,v3)	50(v0,v4,v3)		
v4	30(v0,v4)	30(v0,v4)			
v5	100(v0,v5)	100(v0,v5)	90(v0,v4,v5)	60(v0,v4,v3,v5)	
vj	v2	v4	v3	v5	
S	{v0,v2}	{v0,v2,v4}	{v0,v2,v3,v4}	{v0,v2,v3,v4,v5}	

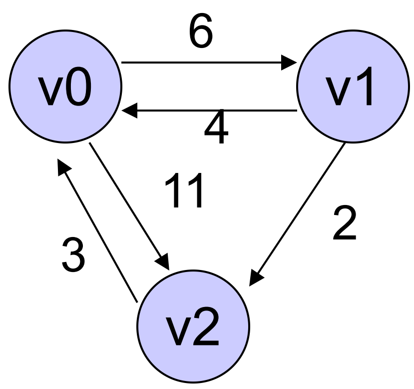
- 7.6.2 每一对顶点之间的最短路径
  - 方法1：调用前面的Dijsktra算法n次
  - 方法2：利用Floyd算法
  - Folyd算法本质上是一个动态规划算法
  - A(i,j): 表示从顶点i到顶点j的最短路径
  - A(i,j)可以由下面状态转移方程决定：

$$A(i,j) = \min_{k\in V}\{A(i,k) + A(k,j), A(i,j)\}$$

- A(i,j)不能通过前面的状态转移方程直接计算。为了计算A(i,j)，我们需要修改状态定义和状态转移方程。
- $A^k(i,j)$:表示从i到j中不经过索引比k大的点的最短路径。
- 新的状态转移方程：

$$A^k(i,j) = \min\{A^{k-1}(i,k) + A^{k-1}(k,j), A^{k-1}(i,j)\}$$

- 显然，$A(i,j) = A^n(i,j)$，其中n是顶点的数目



$$\begin{bmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & \infty & 0 \end{bmatrix}$$

D	$D^{(-1)}$			$D^{(0)}$			$D^{(1)}$			$D^{(2)}$		
	0	1	2	0	1	2	0	1	2	0	1	2
0	0	4	11	0	4	11	0	4	6	0	4	6
1	6	0	2	6	0	2	6	0	2	5	0	2
2	3	∞	0	3	7	0	3	7	0	3	7	0
P	$P^{(-1)}$			$P^{(0)}$			$P^{(1)}$			$P^{(2)}$		
	0	1	2	0	1	2	0	1	2	0	1	2
0		AB	AC		AB	AC		AB	ABC		AB	ABC
1	BA		BC	BA		BC	BA		BC	BCA		BC
2	CA			CA	CAB		CA	CAB		CA		

1. 熟悉图的各种存储结构及其构造算法，了解实际问题的求解效率与采用何种存储结构和算法有密切联系。

2. 熟练掌握图的两种搜索路径的遍历：遍历的逻辑定义、深度优先搜索和广度优先搜索的算法。

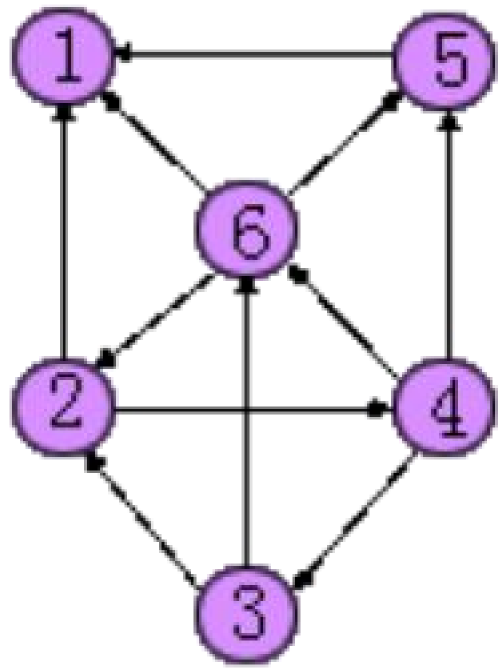
在学习中应注意图的遍历算法与树的遍历算法之间的类似和差异。

3．应用图的遍历算法求解各种简单路径问题。

4．理解教科书中讨论的各种图的算法。

作业

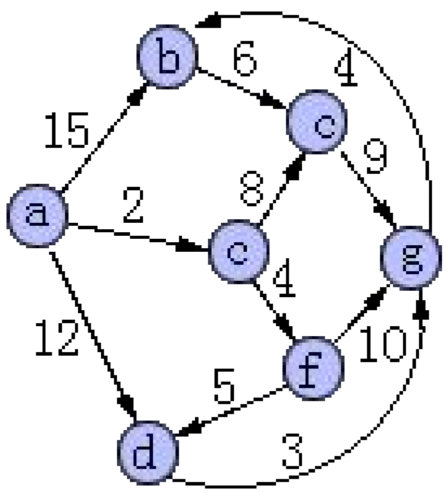
- 1．已知如右图所示的有向图，请给出该图的
  (1) 每个顶点的入/出度；
  (2) 邻接矩阵；
  (3) 邻接表；
  (4) 逆邻接表；
  (5) 强连通分量。



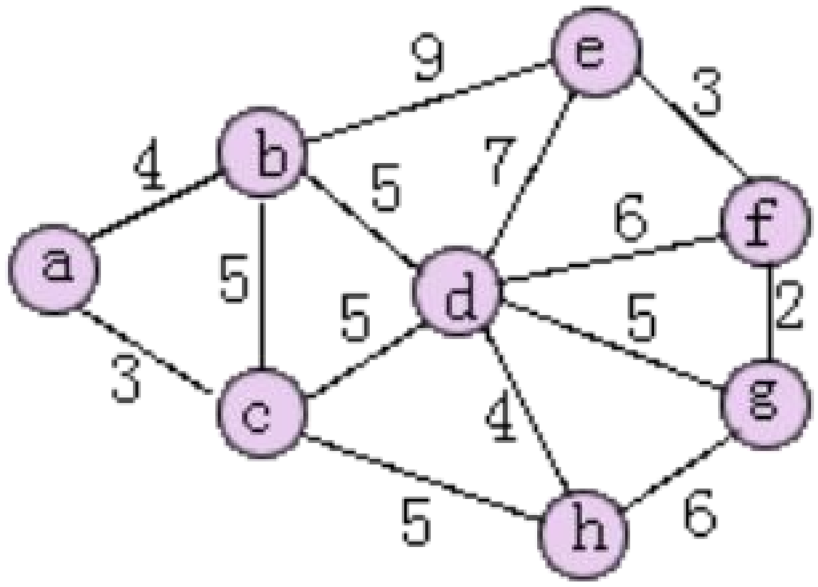
2． 已知以二维数组表示的图的邻接矩阵如右图所示。试分别画出自顶点出发进行遍历所得的深度优先生成树和广度优先生成树。

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

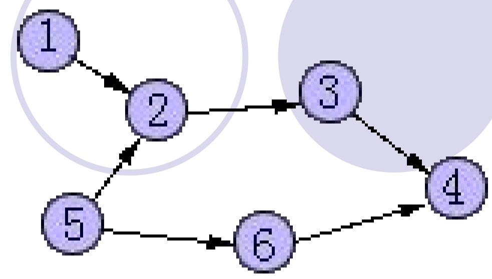
3. 试利用 Dijkstra 算法求右图中从顶点 a 到其它各顶点间的最短路径，写出执行算法过程中各步的状态。



- 4. 请对右边的无向带权图，
  (1) 写出它的邻接矩阵，并按普里姆算法求其最小生成树；
  (2) 写出它的邻接表，并按克鲁斯卡尔算法求其最小生成树。



- 5.试列出右图中全部可能的拓扑有序序列。



- 6.对于右图所示的 AOE 网络，计算各活动弧的 $e(a_i)$ 和 $l(a_j)$ 函数值，各时间(顶点)的 $ve(v_i)$ 和 $vl(v_j)$ 函数值；列出各条关键路径。