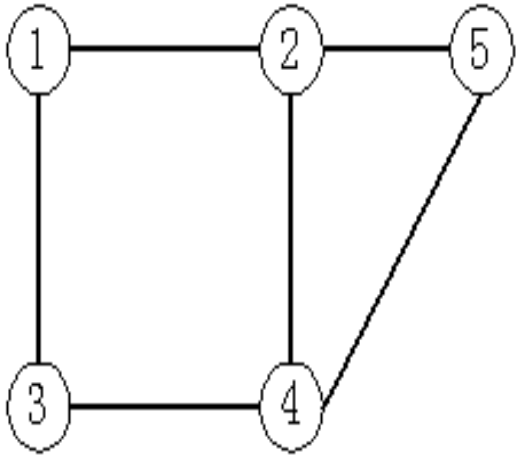
6 图遍历
Graph Traversal

邻接矩阵



G

$$\begin{array}{c} \\ v1 \\ v2 \\ v3 \\ v4 \\ v5 \end{array} \begin{array}{c} \begin{array}{ccccc} v1 & v2 & v3 & v4 & v5 \end{array} \\ \begin{pmatrix} a11 & a12 & a13 & a14 & a15 \\ a21 & a22 & a23 & a24 & a25 \\ a31 & a32 & a33 & a34 & a35 \\ a41 & a42 & a43 & a44 & a45 \\ a51 & a52 & a53 & a54 & a55 \end{pmatrix} \end{array}$$

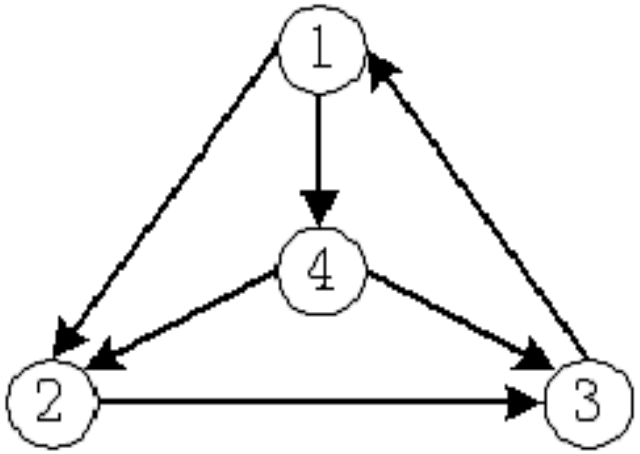
构造五行五列的方阵

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

图G对应的方阵

其中各结点的表示:v1=1, v2=2, v3=3, v4=4, v5=5

邻接矩阵（有向图）



G

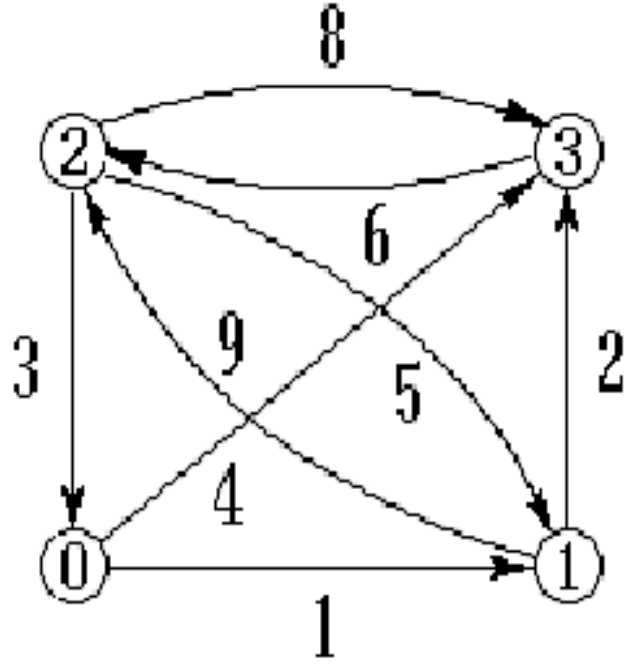
$$\begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

有向图G的邻接矩阵

有向图的邻接矩阵可能是不对称的。在有向图中：

- **▲ 第 *i* 行中 1 的个数就是顶点 *i* 的出度。**
- **▲ 第 *j* 列 中1 的个数就是顶点 *j* 的入度。**
- **▲ 有向图中各顶点的入度之和等于出度之和。**

带权值的邻接矩阵



$$\begin{pmatrix} 0 & 1 & \infty & 4 \\ \infty & 0 & 9 & 2 \\ 3 & 5 & 0 & 8 \\ \infty & \infty & 6 & 0 \end{pmatrix}$$

$$a[i][j] = \begin{cases} W(i,j), & \text{如果 } i != j \text{ 且 } <i,j> \in E \text{ 或 } (i,j) \in E \\ \infty, & \text{否则, 但是 } i != j \\ 0, & \text{对角线 } i == j \end{cases}$$

w(i, j)表示顶点i和顶点j之间边的权值。

无向图的邻接表(不带权)

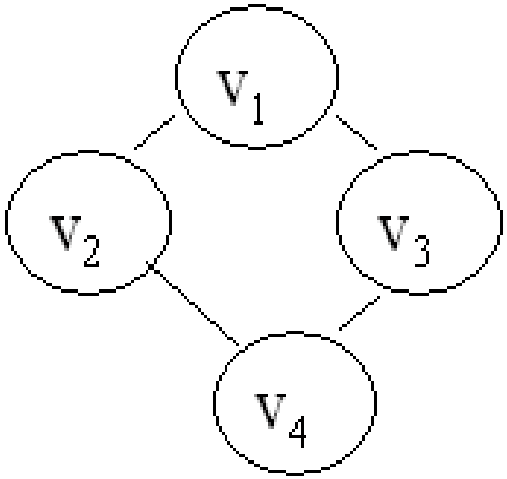
- 每个链表的前边附设一个表头结点。
- 把同一个顶点发出的边链接在同一个边链表中，链表的每一个结点代表一条边，叫做表结点。

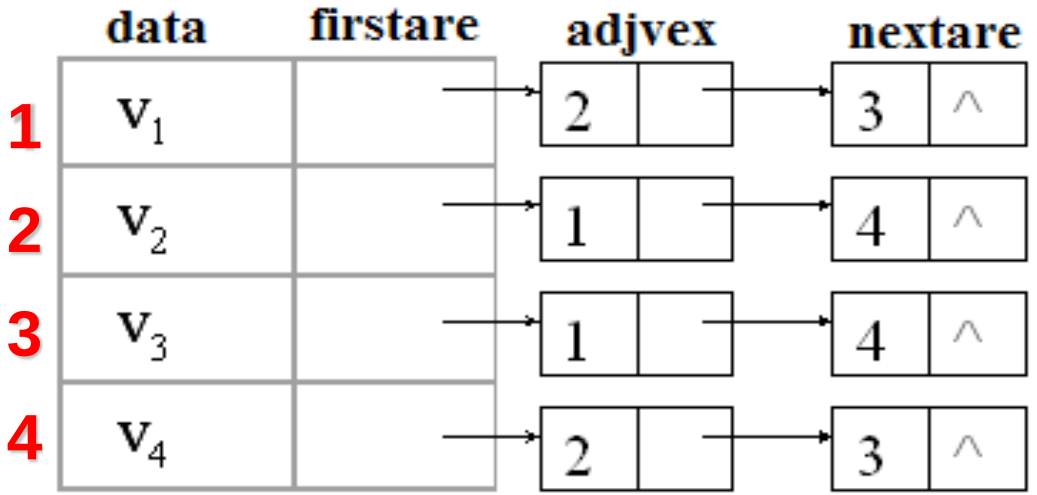
data	firstare

表头结点

adjvex	nextare

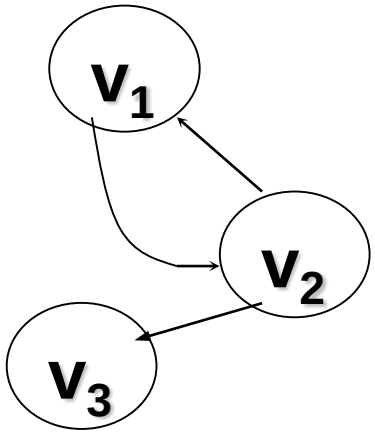
表结点



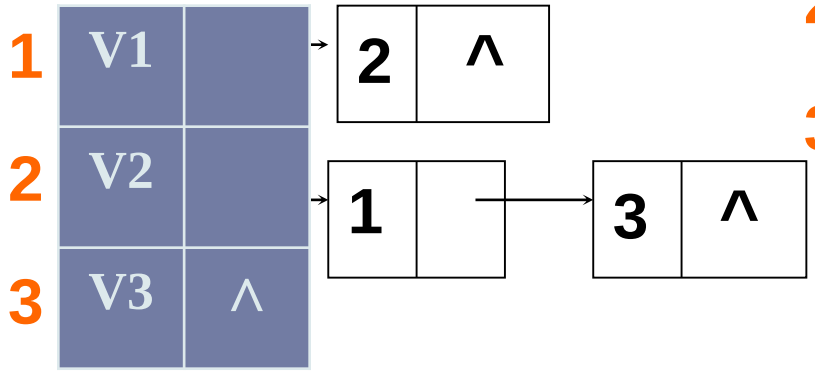


假设数组下标自1开始

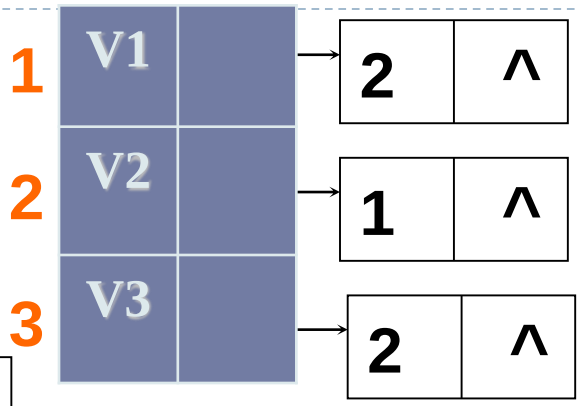
有向图的邻接表和逆邻接表



G



邻接表



逆邻接表

- 在有向图的邻接表中，第 *i* 个边链表链接的边都是顶点 *i* 发出的边。也叫做出边表。
- 在有向图的逆邻接表中，第 *i* 个边链表链接的边都是进入顶点 *i* 的边。也叫做入边表。

网络 (带权图) 的邻接表

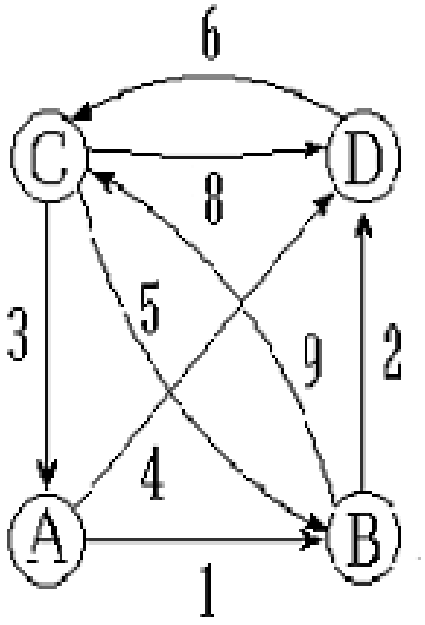
边表结点

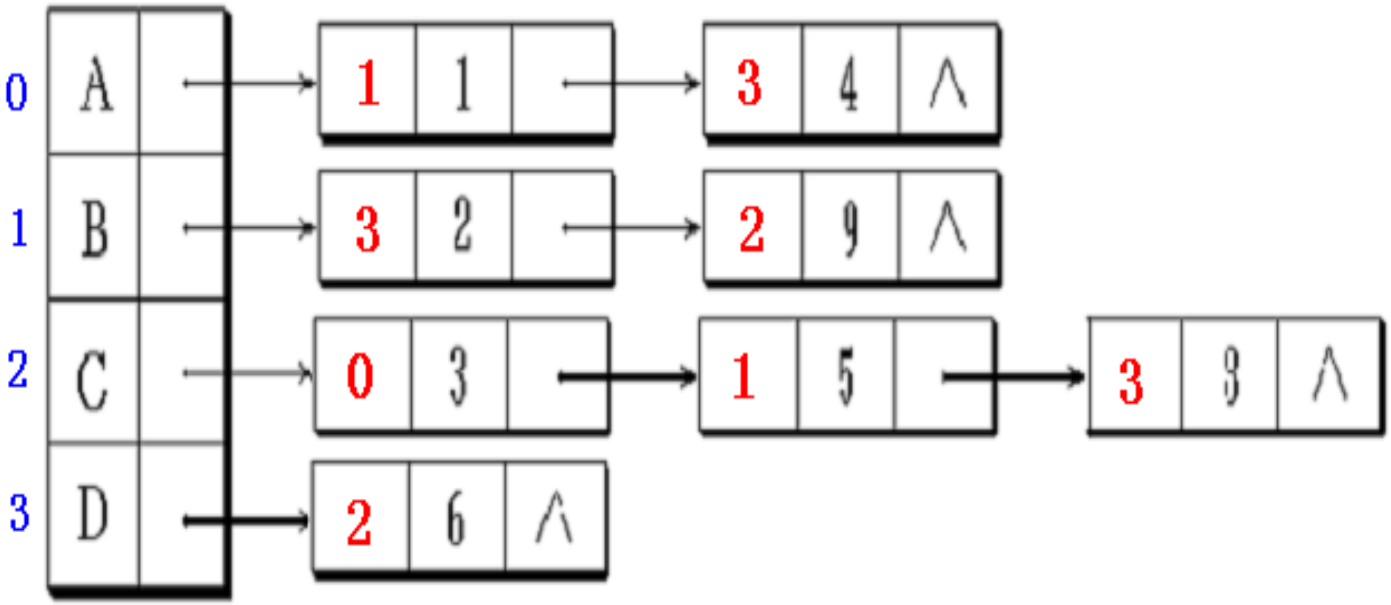
adjvex	info	nextare

adjvex：顶点号

info：边上所带的权

nextare：指针





十字链表——适用于有向图

它是有向图的另一种链式结构。

思路：将邻接矩阵用链表存储，是邻接表、逆邻接表的结合。

1、每条弧对应一个结点(称为弧结点，设5个域)

2、每个顶点也对应一个结点（称为顶点结点，设3个域）

顶点结点

data	Firstin	Firstout

data ：存储顶点信息。
Firstin ：以顶点为弧头的第一条弧结点。
Firstout: 以顶点为弧尾的第一条弧结点。

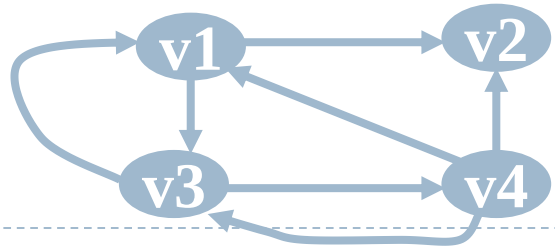
弧结点

tailvex	headvex	Hlink	tlink	info

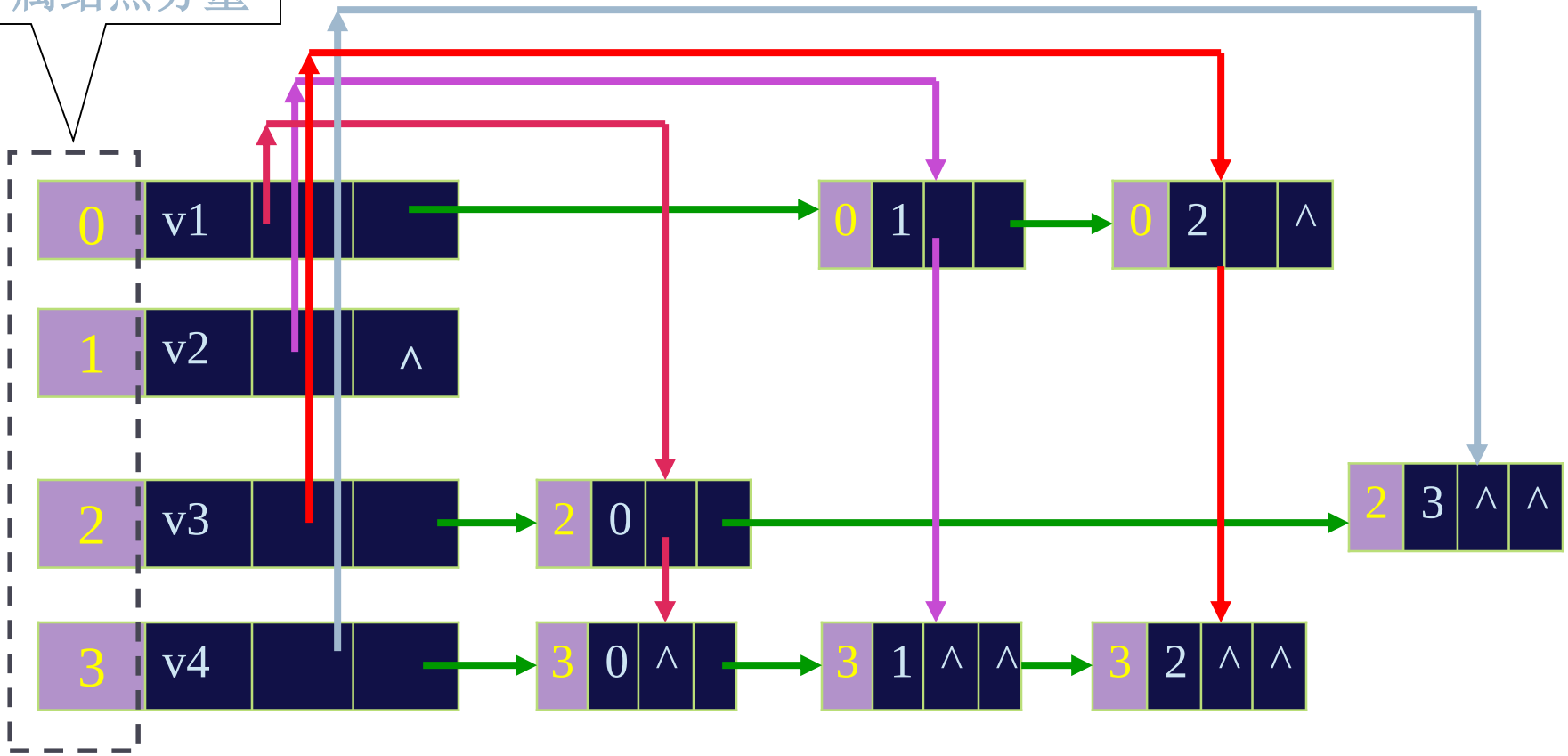
tailvex: 弧尾顶点位置
headvex: 弧头顶点位置
hlink: 弧头相同的下一弧位置
tlink: 弧尾相同的下一弧位置
info: 弧信息

n个顶点—用顺序存储结构

例：画出有向图的十字链表。



数组下标不属结点分量

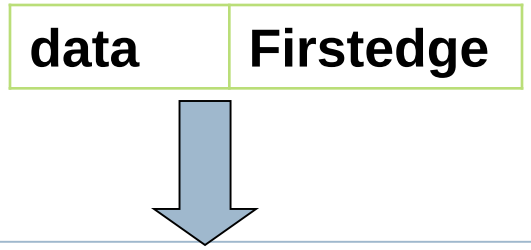


十字链表优点：容易操作，如求顶点的入度、出度等。

空间复杂度与邻接表相同；建立的时间复杂度与邻接表相同。

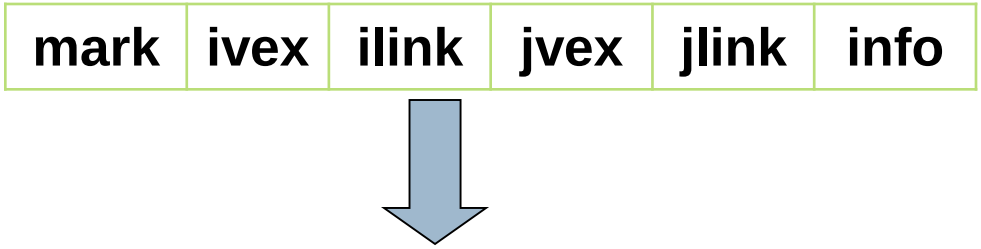
邻接多重表——适用于无向图

顶点结点



data ： 存储顶点信息。
Firstedge ： 依附顶点的第一条边结点。

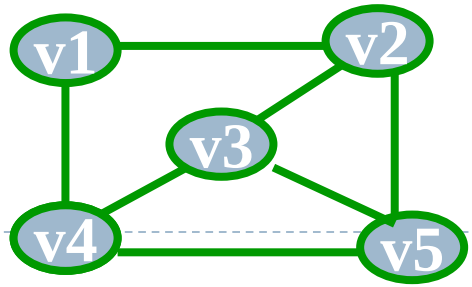
边结点



mark：标志域，如处理过或搜索过。
ivex，jvex ： 顶点域，边依附的两个顶点位置
ilink： 指向下一条依附顶点 i 的边结点位置。
Jlink: 指向下一条依附顶点 j 的边结点位置。
info: 边信息，如权值等。

n个顶点——用顺序存储结构

例：画出无向图的邻接多重表



邻接多重表优点：

容易操作，如求顶点的度等。

空间复杂度与邻接表相同；

建立的时间复杂度与邻接表相同。

- 图的遍历是求解图问题的基础。
- 和树的遍历类似，图的遍历希望从图中某一顶点出发，对其余各个顶点都访问一次，但比树的遍历要复杂得多。
- 图的任一顶点都有可能和其余顶点相邻接，因此在访问了某顶点后，可能沿着某条路径搜索以后，又回到该顶点。
- 通常有两种遍历图的方法：深度优先搜索、广度(宽度)优先搜索。他们都适合于无向图和有向图。
- 本章重点内容是介绍两种图遍历算法，并学习图遍历算法的一些应用。

图的两种遍历方法

- 深度优先搜索(Depth-First Search, DFS)
- 宽度优先搜索(Breadth-First Search, BFS)

深度优先搜索(Depth-First Search, DFS)

给定有向或是无向图G = (V,E)，DFS工作过程如下：

1. 将所有的顶点标记为”unvisited”。
2. 选择一个起始顶点,不妨称为v ∈V,并将之标记为”visited”。
3. 选择与v相邻的任一顶点，不妨称之为w，将w标记为”visited”。
4. 继续选择一个与w相邻且未被访问的顶点作为x；将x标记为”visited”。继续选择与x相邻且未被访问的顶点。
5. 此过程一直进行，直到发现一个顶点y，邻接于y的所有顶点都已经被标记为”visited”。此时，返回到最近访问的顶点，不妨称之为z，然后访问和z相邻且标记为”unvisited”的顶点。
6. 上述过程一直进行，直到返回到起始顶点v。

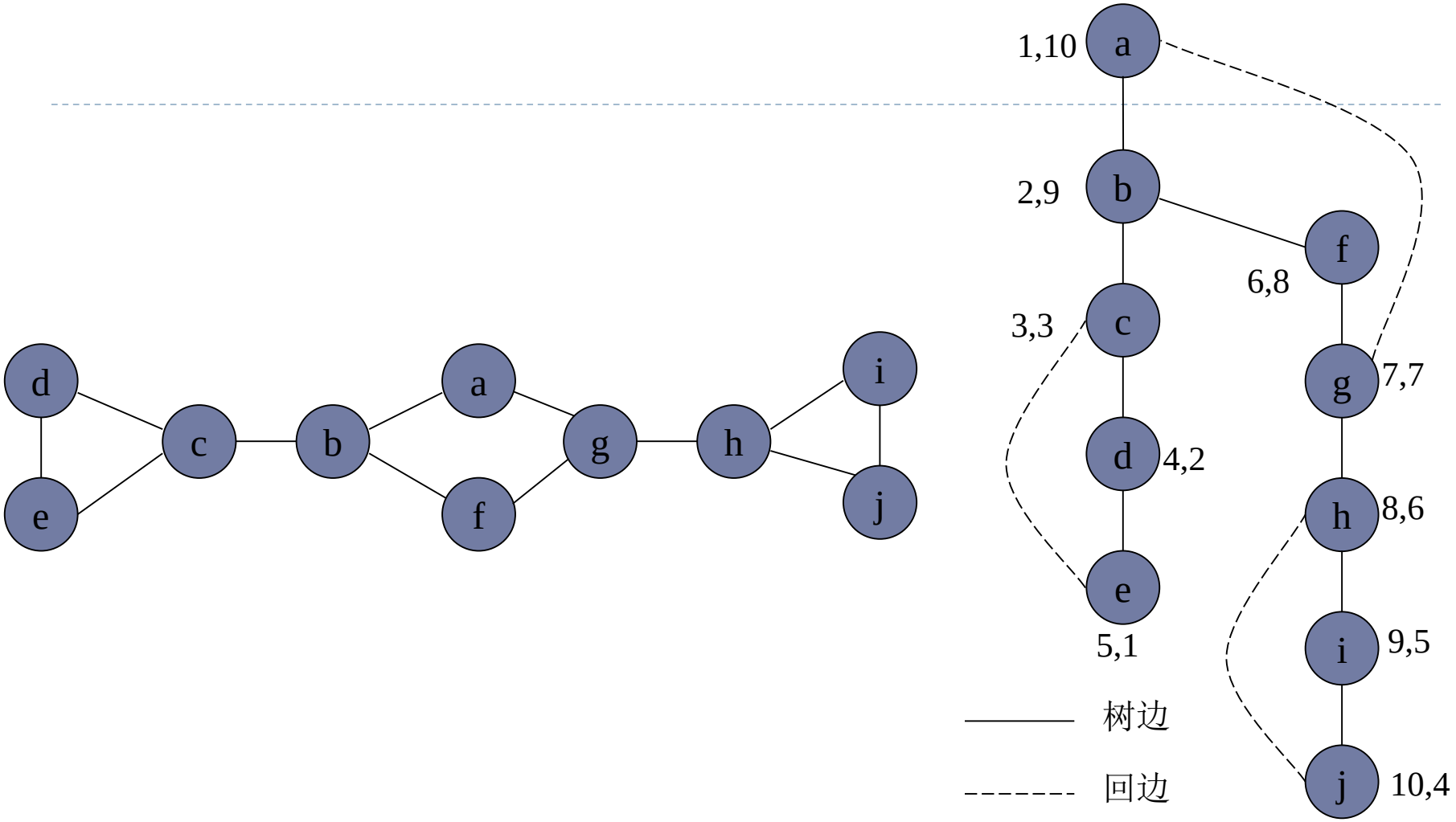
- 深度优先搜索生成树(depth-first search spanning tree)
- 深度优先搜索生成森林(depth-first search spanning forest)（起始点出发不是所有的顶点都可到达）
- predfn：在图的深度优先搜索生成树(森林)中顶点的先序号。所谓先序号，是指按照先序方式访问该生成树，该顶点的序号。
- postdfn：在图的深度优先搜索生成树(森林)中顶点的后序号。所谓后序号，是指按照后序方式访问该生成树，该顶点的序号。
- 对边(v,w)进行探测的含义是：检查(v,w)以测试w是否已经被访问过(”visited”)。

输入：无向图或有向图G= (V,E)
输出：深度优先搜索树(森林)中每个顶点的先序号、后序号

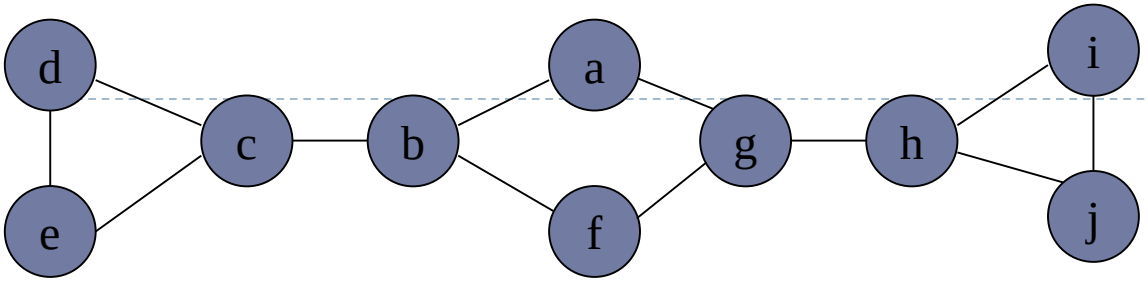
```
1. predfn←0; postdfn←0  //计数器，在使用DFS解决某些实际问题时用到
2. for v∈V
3.   visited[v] ← false
4. end for
5. for v∈V     //从一个顶点v出发，可能无法遍历全部顶点
6.   if visited[v] = false  then dfs(v)
7. end for

dfs(v)
1.     visited[v] ←true
2.     predfn←predfn+1
3.     for (v,w)∈E
4.      if visited[w]=false then dfs(w)
5.     end for
6.     postdfn←postdfn+1.
```

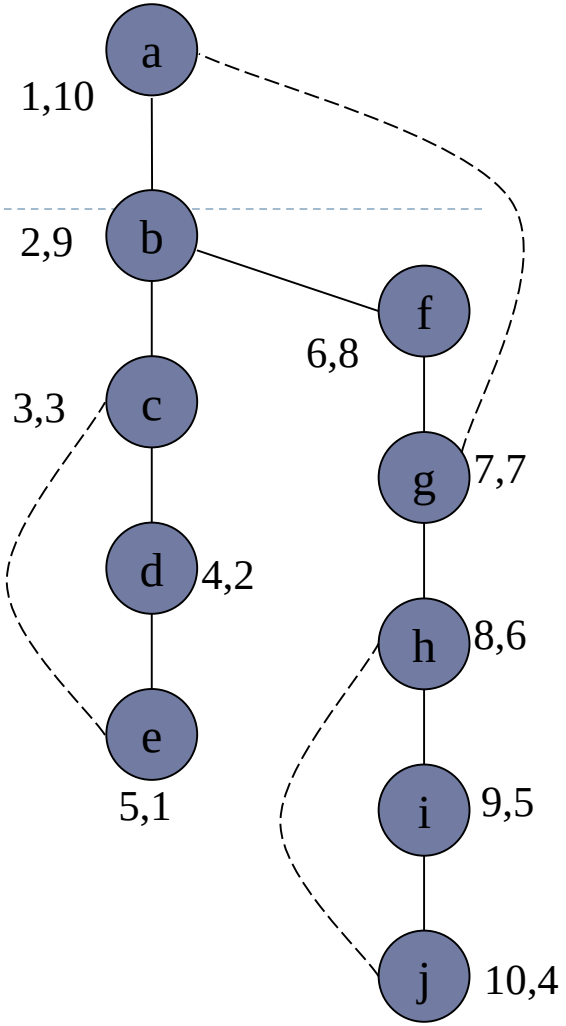
无向图情形

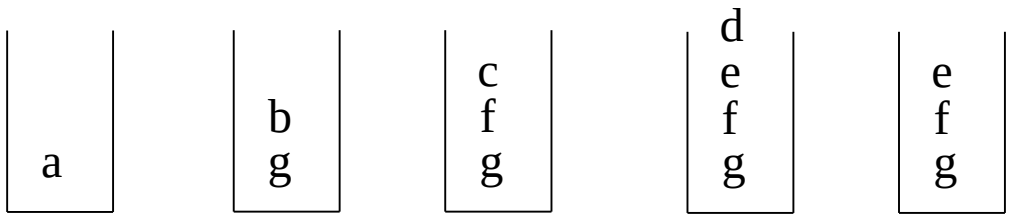
- 无向图G= (V,E)，深度优先遍历后，G中的边可以分为如下类型：
- 树边(Tree edges) – 深度优先搜索生成树中的边。具有如下性质：探测边(v,w)时，w是“unvisited”状态，则边(v,w)是树边。
- 回边(Back edges) – G中除了树边的所有其它边。

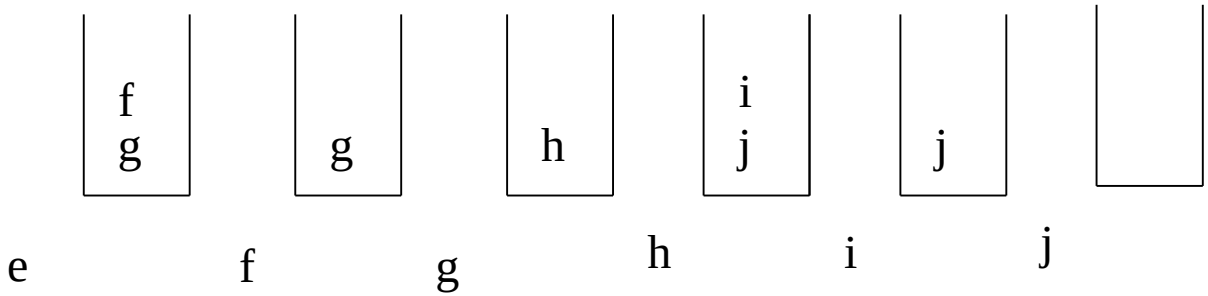


postdfn=1: 该顶点是第一个不能继续向深度前进的顶点；
或是称为第一个完成深度搜索的顶点。









使用栈的方式来实现DFS

借助一个堆栈实现迭代形式的DFS

```
dfs(v)
1.    visited[v] ←true
2.    predfn←predfn+1
3.    for (v,w)∈E  //总共的循环次数
4.        if visited[w]=false then dfs(w)
5.    end for
6.    postdfn←postdfn+1.
```

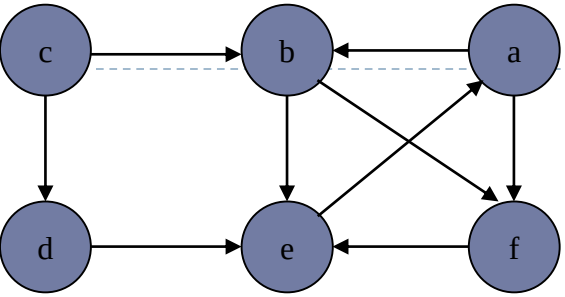
递规形式

```
dfs(v)
1.    Push(v,S)
2.    visited[v] ←true
3.    While S!={}
4.        v ←Pop(S)  //visit
5.        for (v,w)∈E
6.          if visited[w]=false then
7.                Push(w,S)
8.                visited[w] = true
9.            end if
10.       end for
11.   end while
```

迭代形式

有向图情形

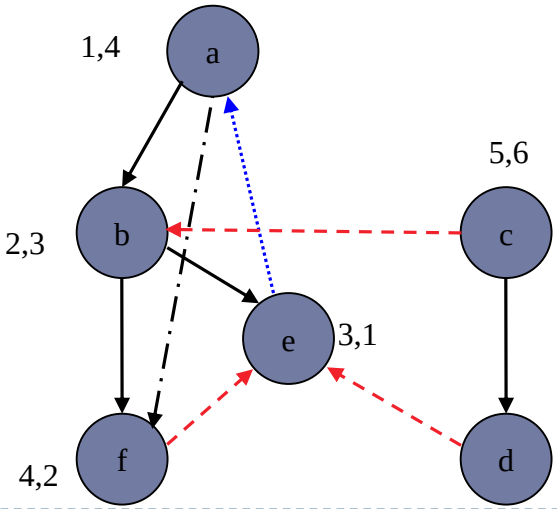
- 有向图G= (V,E)，深度优先遍历后，会生成一个或是多个(有向)深度优先搜索生成树。G中的边可以分为如下4种类型：
- 树边(Tree edges) – 深度优先搜索生成树中的边：探测边(v,w)时，w是“unvisited”状态，则边(v,w)是树边。
- 回边(Back edges) – 在迄今为止所构建的深度优先搜索生成树中，w是v的祖先，并且在探测(v,w)时，w已经被标记为”visited”,则(v,w)为回边。
- 前向边(Forward edges) – 在迄今为止所构建的深度优先搜索生成树中，w是v的后裔，并且在探测(v,w)时，w已经被标记为”visited”,则(v,w)为前向边。
- 横跨边(Cross edges) – 所有其他的边。



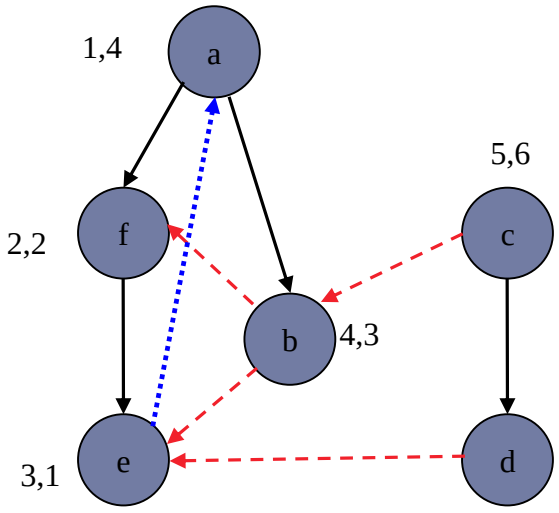
- 回边(Back edges)－在迄今为止所构建的深度优先搜索生成树中，w是v的祖先，并且在探测(v,w)时，w已经被标记为”visited”，则(v,w)为回边。
- 前向边(Forward edges)－在迄今为止所构建的深度优先搜索生成树中，w是v的后裔，并且在探测(v,w)时，w已经被标记为”visited”，则(v,w)为前向边。

树边　　前向边　　回边　　横跨边

Case 1: 从顶点a开始，
依次访问a,b,e,f; c,d



Case 2: 从顶点a开始，
依次访问a,f,e,b; c,d



为便于分析深度优先搜索算法的执行情况，在搜索中为每个结点添加**时间戳**。

深度优先搜索对每个结点u加盖**两个时间戳**d[u]和f[u]：

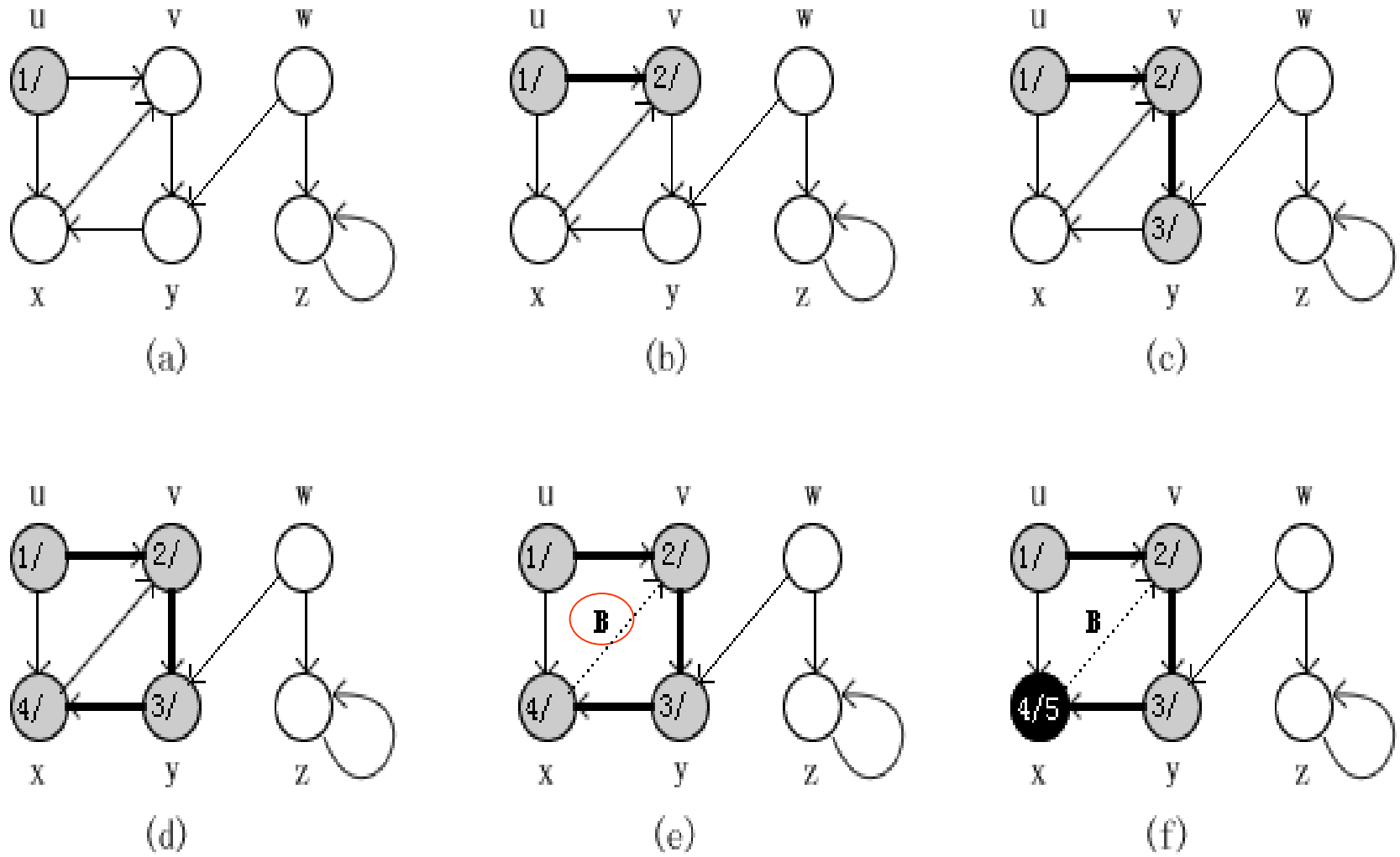
第1个时间：当结点u着灰色时，由d[u]记录；

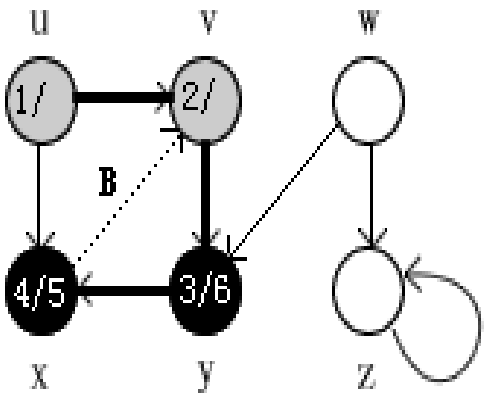
第2个时间：当结点u着黑色时，由f[u]记录。

在时刻d[u]之前结点u为白色，在时刻d[u]和f[u]之间为灰色，f[u]以后变为黑色。

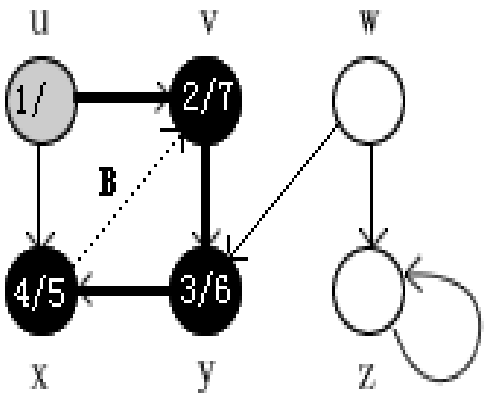
初始时，对所有结点u，d[u] = f[u]=0，时钟的初值为0.

图2说明了深度优先搜索在有向图（u出发）上的执行过程

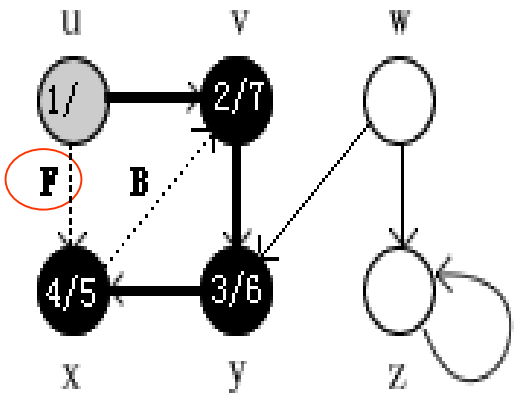



u
v
w
1/
2/
B
4/5
3/6
x
y
z

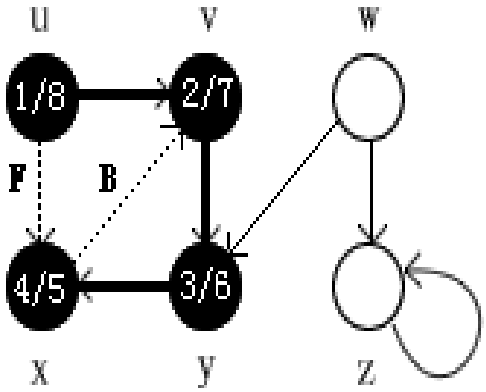
(g)


u
v
w
1/
2/7
B
4/5
3/6
x
y
z

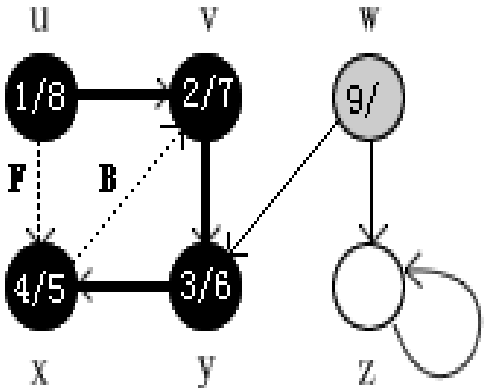
(h)


u
v
w
1/
2/7
F
B
4/5
3/6
x
y
z

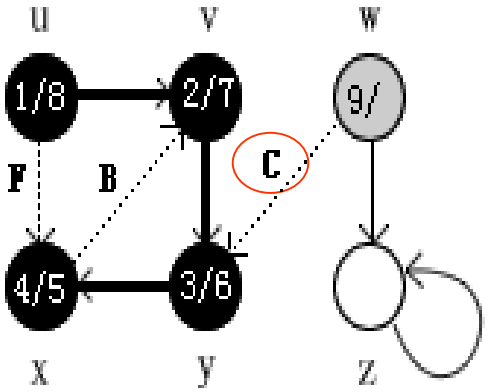
(i)


u
v
w
1/8
2/7
F
B
4/5
3/6
x
y
z

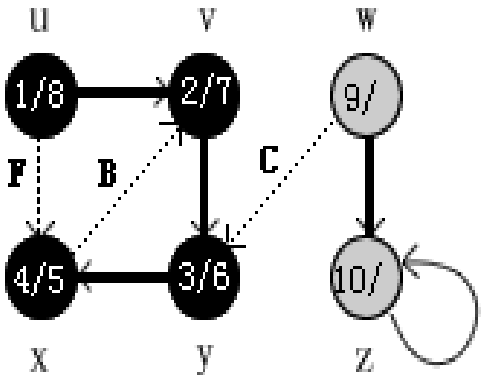
(j)


u
v
w
1/8
2/7
9/
F
B
4/5
3/6
x
y
z

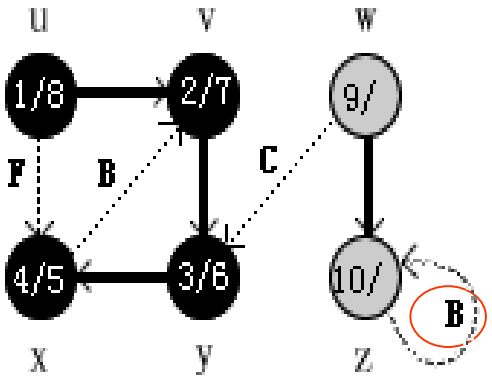
(k)


u
v
w
1/8
2/7
9/
C
F
B
4/5
3/6
x
y
z

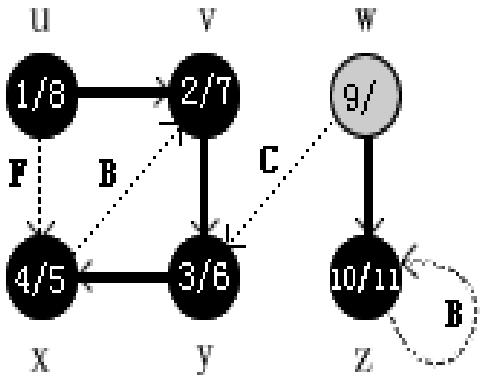
(l)



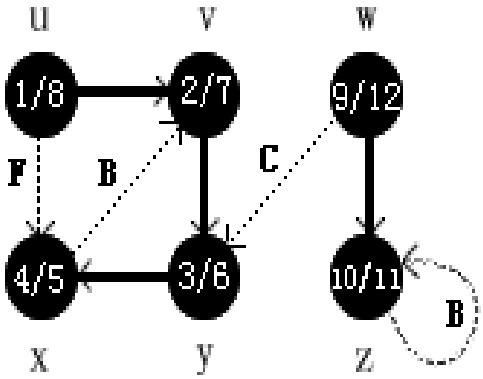
(m)



(n)



(o)



(p)

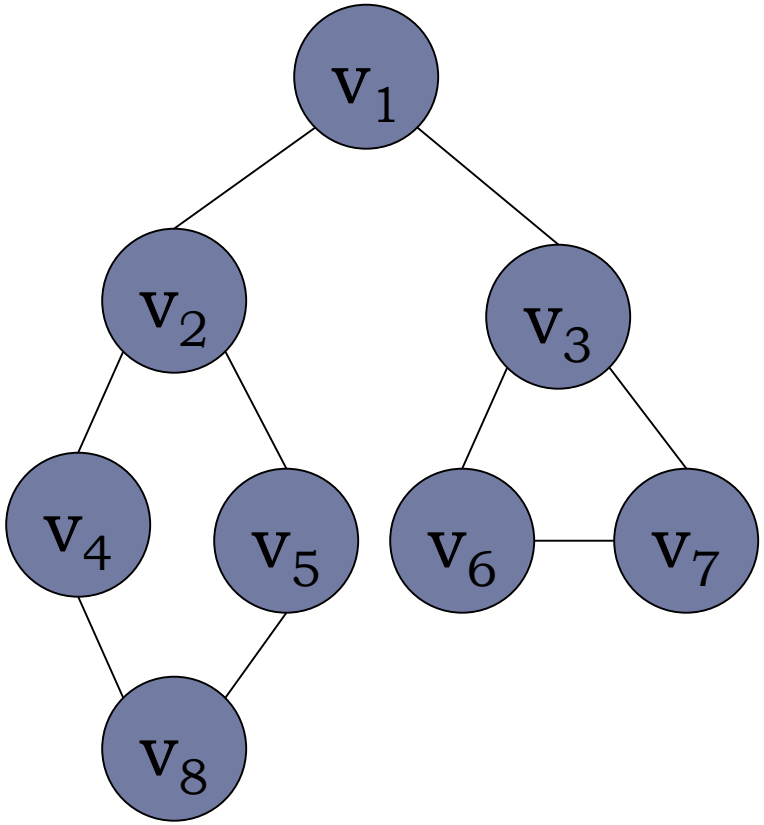
- 加粗的实线边--树枝
- 虚线边--非树枝的边，分别标明B、C、F以表示回边、横跨边、抢险边。

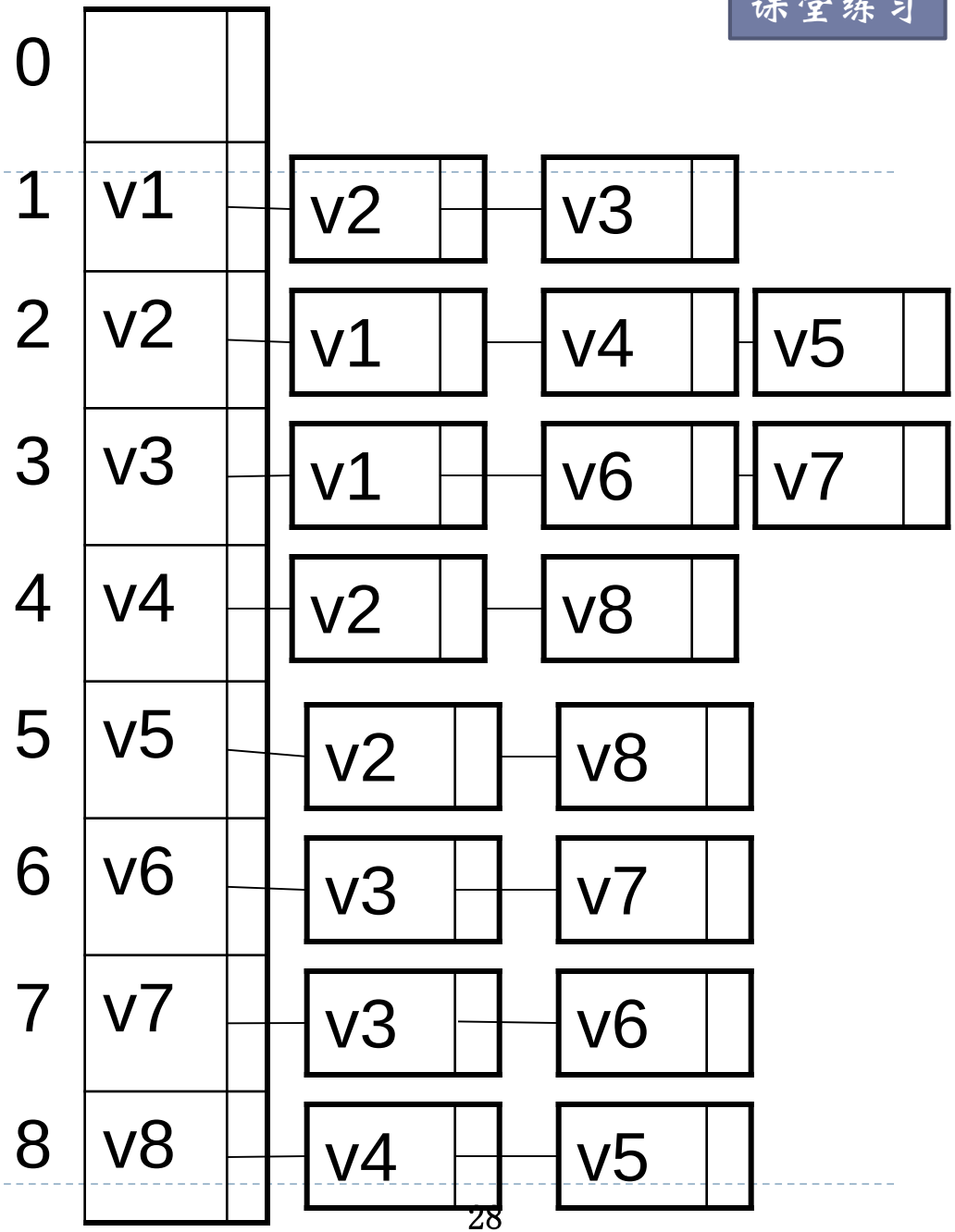
为何DFS用于无向图时，不存在前向边?

- 树边(Tree edges) – 深度优先搜索生成树中的边：探测边(v,w)时，w是“unvisited”状态，则边(v,w)是树边。
- 回边(Back edges) – 在迄今为止所构建的深度优先搜索生成树中，w是v的祖先，并且在探测(v,w)时，w已经被标记为”visited”，则(v,w)为回边。
- 前向边(Forward edges) – 在迄今为止所构建的深度优先搜索生成树中，w是v的后裔，并且在探测(v,w)时，w已经被标记为”visited”，则(v,w)为前向边。
- 横跨边(Cross edges) – 所有其他的边。

假设存在前向边，则由前向边定义可设w是v的后裔，且探索(v,w), w已被访问。由于G为无向图，故(v,w)即(w,v)，当深度优先搜索 树构建到w时，首先考虑由w出发的边，故(w,v)在(v,w)之前被探索， 则该边被记为回边，而非前向边。

对于下图及其邻接表，写出以v_2、v_8为起始点的深度优先遍历序列，并画出深度优先树，标出边的分类。





时间复杂度分析(课后阅读)

- 假设图有n个顶点，m条边。图使用邻接链表存储。
- Θ(m+n)

输入：无向图或有向图G= (V,E)
输出：深度优先搜索树(森林)中每个顶点的先序号、后序号

```
1. predfn←0; postdfn←0  //
2. for v∈V
3.   visited[v] ← false    //  Θ(n)
4. end for
5. for v∈V     //从一个顶点v出发，可能无法遍历全部顶点
6.   if visited[v] = false  then dfs(v)
7. end for

dfs(v)
1.     visited[v] ←true
2.     predfn←predfn+1
3.     for (v,w)∈E  //总共的循环次数 Θ(m)
4.         if visited[w]=false then dfs(w)
5.     end for
6.     postdfn←postdfn+1.
```

- 如果使用邻接矩阵，时间复杂度？

$$\Theta(m) \longrightarrow \Theta(n^2)$$

深度优先搜索的应用

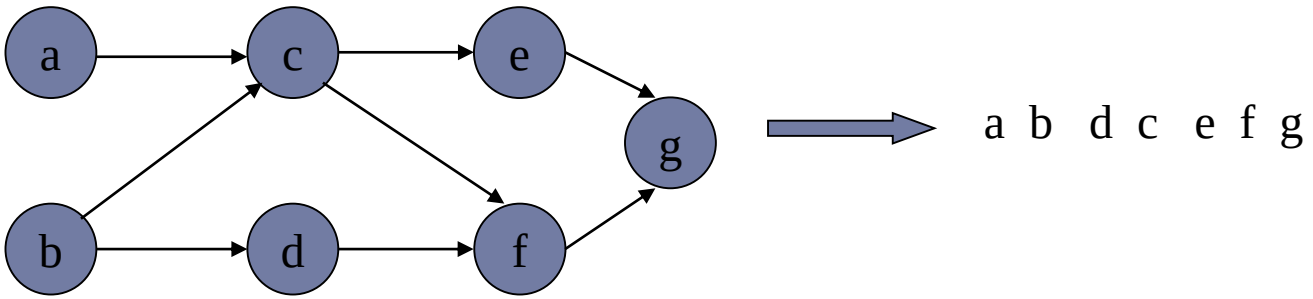
- 图的无回路性判定
- 拓扑排序
- 寻找图的关节点和双连通分量
- 网络页面检索

图回路判定

- 问题：若G= (V,E)为一个有n个顶点和m条边的有向或是无向图。要测试G中是否包含有一个回路。
- 方法：对G施加深度优先搜索，如果探测到一个回边，那么可以判定G中含有回路；否则G中无回路。
- 注意：如果G是连通的无向图，则不需要对G进行深度优先搜索来判定是否有回路。G无回路，当且仅当|E|=|V|-1。

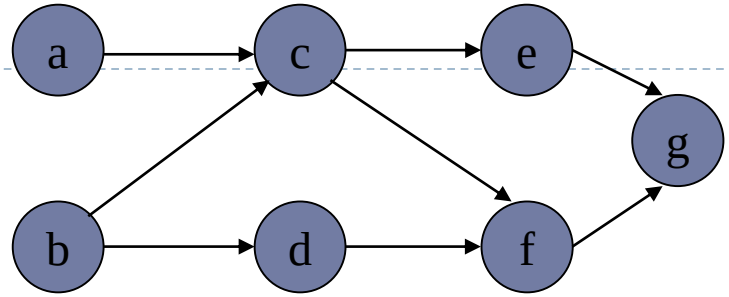
拓扑排序(Topological sorting)

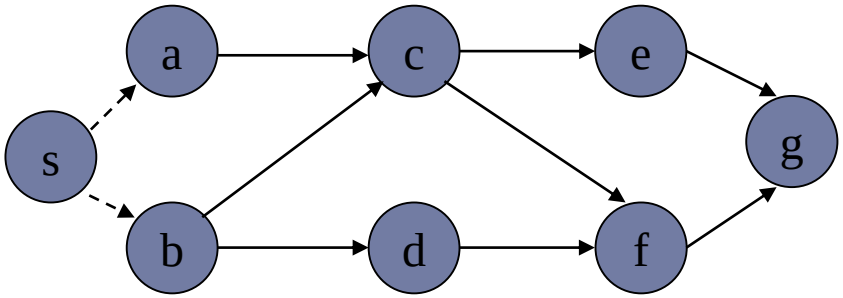
- 给定一个有向无回路图(Directed Acyclic Graph, DAG) G=(V,E)。拓扑排序是为了找到图顶点的一个线性序，使得：如果(v,w) ∈E,那么，在线性序中，v在w之前出现。

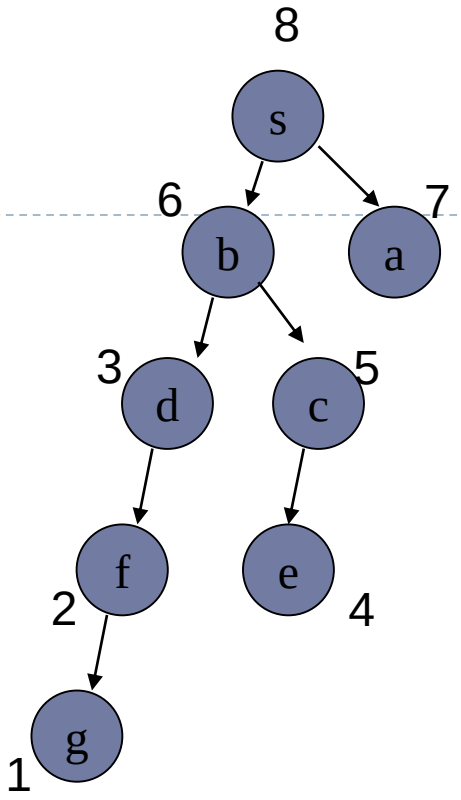


- 我们假设在DAG中只有唯一一个入度为0的顶点；如果有一个以上的顶点入度为0，可以通过添加一个新的顶点s，然后将s指向所有入度为0的顶点，这样s就成为唯一一个入度为0的顶点。

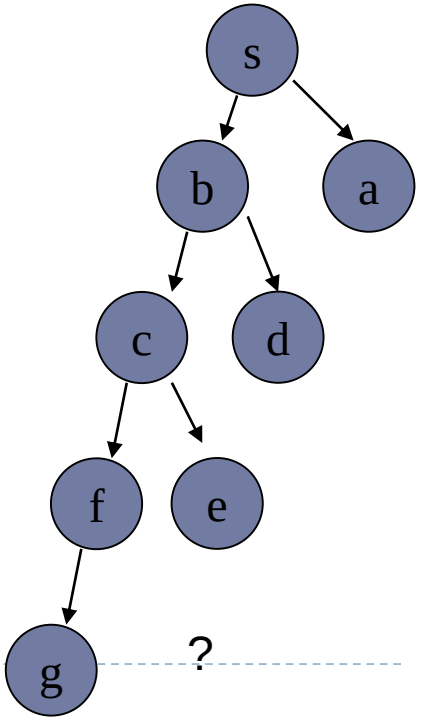
- 拓扑排序的实现：
  - 从入度为0的顶点开始，对DAG实施深度优先搜索。
  - 遍历完成后，计数器postdfn恰好对应于一个在DAG中顶点的反拓扑序。
- 得到拓扑序：
  - 在DFS算法中恰当位置增加输出语句(step 6)，然后将输出结果反转。
  - 在DFS算法中恰当位置增加顶点入栈(step 6)操作，然后依次取栈顶元素输出。







a b c e d f g



4.3 双连通分量

针对无向图

4.3.1 基本概念

双连通图：

一个无向图的任意两个结点之间至少有两条不同的路径相通。

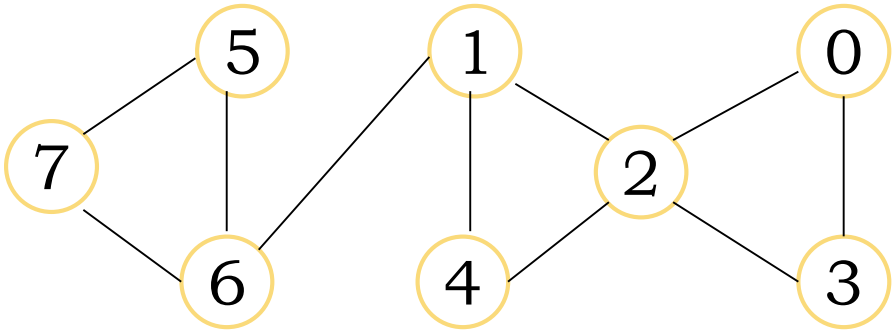
关节点：

在一个无向连通图G=(V,E)中，可能存在某个（或多个）结点a，使得一旦删除a及其相关联的边，图G不再是连通图，则结点a称为图G的关节点。

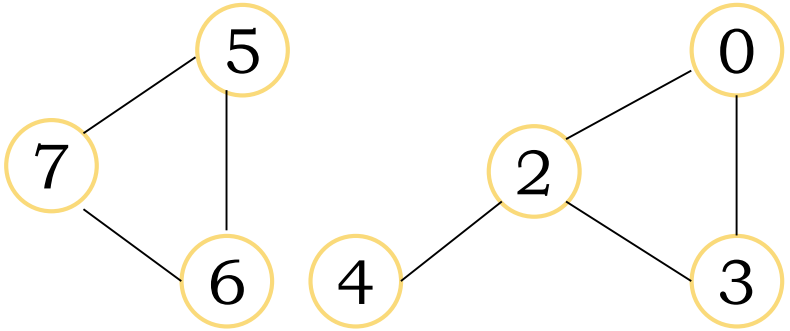
桥：

如果删除图G的某条边b，该图分离成两个非空子图，则称边b是图G的桥。

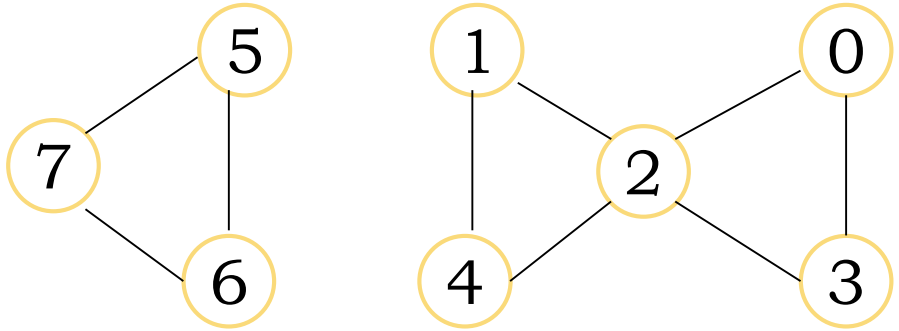
图中的图G有哪些关节点？哪条边是桥？



(a) 无向连通图G

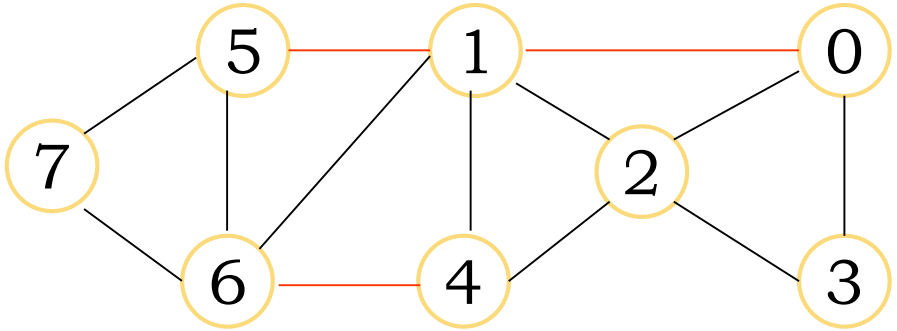


(b) 删除图G的关节点1



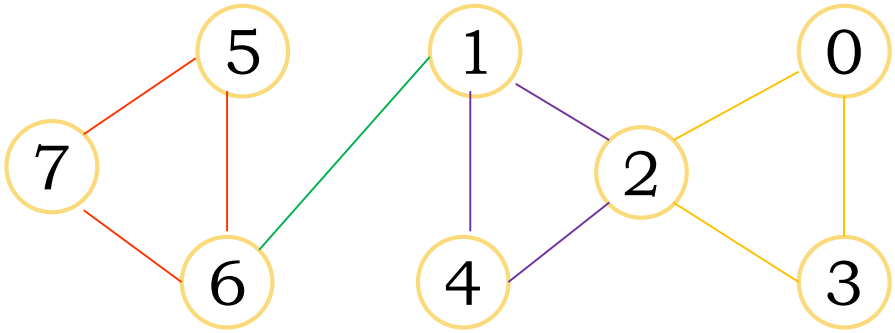
(c) 删除图G的桥<6, 1>

双连通图：无向连通图G中不包含任何关节点。



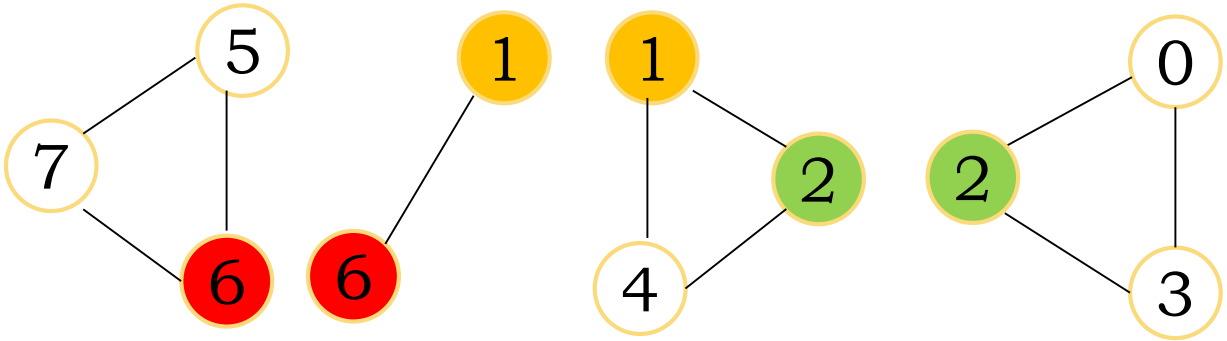
双连通分量：无向连通图G的极大双连通子图。

一个无向图可以分成多个双连通分量，它们将图中的边划分为若干个子集（不是将结点划分子集）。



(a)无向连通图G

从图中可以看出：
1. 两个双连通分量至多有一个公共结点，且此结点必为关节点。
2. 两个双连通分量不可能共有同一条边。
3. 每个双连通分量至少包含两个结点（除非无向图只有一个结点）



双连通分量

对于一个无向连通图G，下列说法是等价的：

（1）图G是双连通的；

（2）图G的任意两个结点之间存在简单回路；

（3）图G中不包含关节点。

简单回路：在一个回路中，出现的边互不相同。

发现关节点

- 在网络应用中，通常不希望网络中存在关节点。因为这意味着一旦在这些位置出现故障，势必导致大面积的通信中断。
- 因此判定一个无向图是否双连通图，在图中发现关节点及求图的双连通分量是很有实际意义的问题。

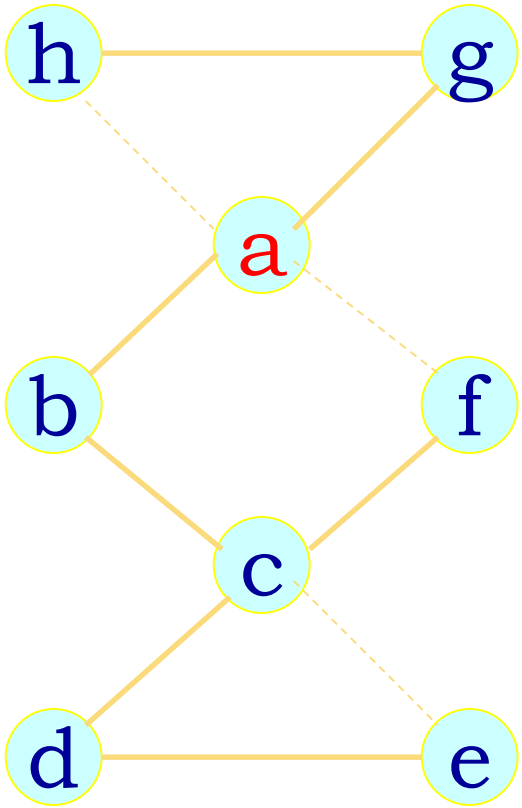
一个无向连通图不是双连通图的充要条件是图中存在关节点。

在无向图中识别关节点的最简单做法是：从图G中删除一个结点a和该结点的关联边，再检查图G的连通性。如果图G因此而不再是连通图，则结点a是关节点。

需借助图的深度优先生成树来识别关节点。

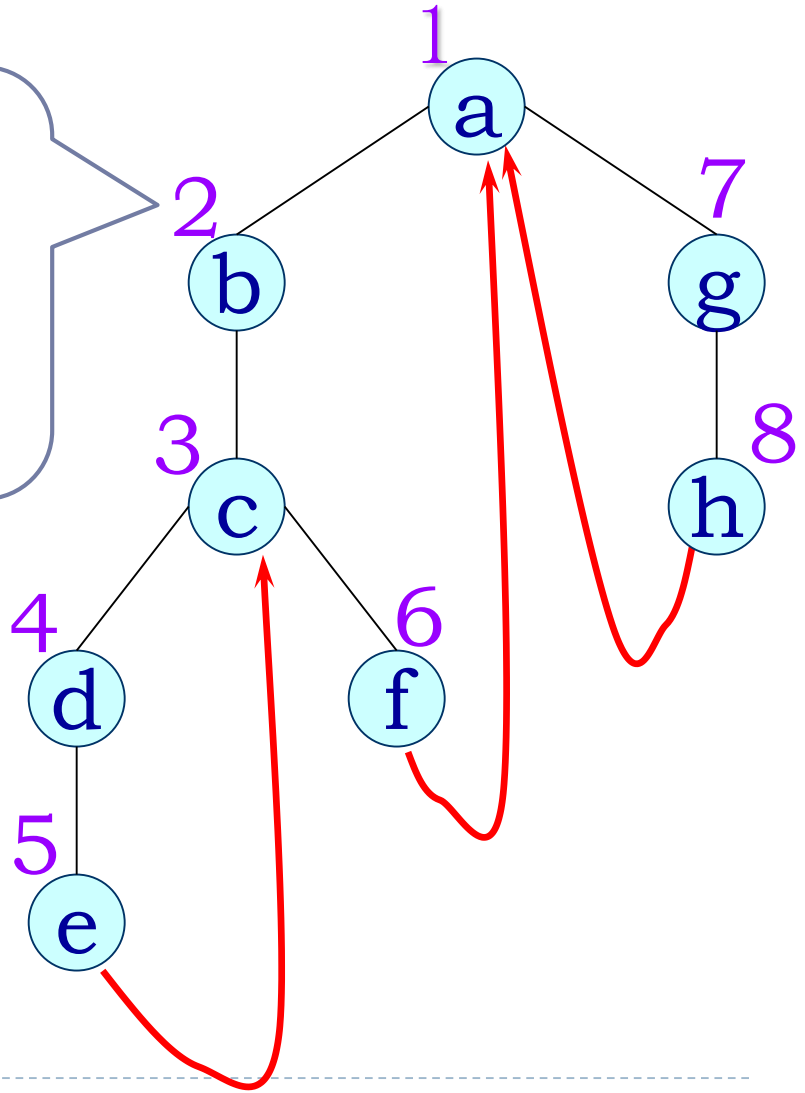
假设从某个顶点V_0出发对连通图进行深度优先搜索，则可得到一棵深度优先生成树，树上包含图的所有顶点。

例如:下列连通图中的关节点?



结点的深度优先数，记录了结点被访问的先后次序

深度优先生成树(a为根结点)



结点a,c是关节点

性质　给定无向连通图G=(V,E),S=(V,T)是图G的一棵深度优先树，图中结点a是一个关节点，当且仅当

（1）a是根，且a至少有两个孩子。

（2）或者a不是根，且a的某棵子树上没有指向a的祖先的回边。

求图G关节点的算法

深度优先数：在深度优先搜索中对每个结点u加盖两个时间戳。其中，d[u]记录结点u被访问的时间，也称为结点的深度优先数；

为了实现这一算法，需要对图中每个结点定义一个与优先数有关的量Low。

最低深度优先数：Low[u]表示从结点u出发，经过某条路径可以达到深度优先树其他结点的最小优先数；

从结点u出发，有两种途径可以达到树中其他结点：

一、自结点u经过一条回边到达某个结点x；

二、自结点u出发，经过u的某个孩子w，以及一条由w出发的由树边组成的一条路径和回边到达某个结点y；

Low[u]是结点u通过一条子孙路径和至多随后一条回边所能到达的结点的最低深度优先数。

定义：Low[u]定义如下：

Low[u]=min{ d[u],

min{ Low[w] | w是u的孩子},

min{ d[x] | (u,x)是一条回边} }

如果u不是根，当且仅当结点u有一个孩子w，其Low[w]≥d[u]时，u是一个关节点

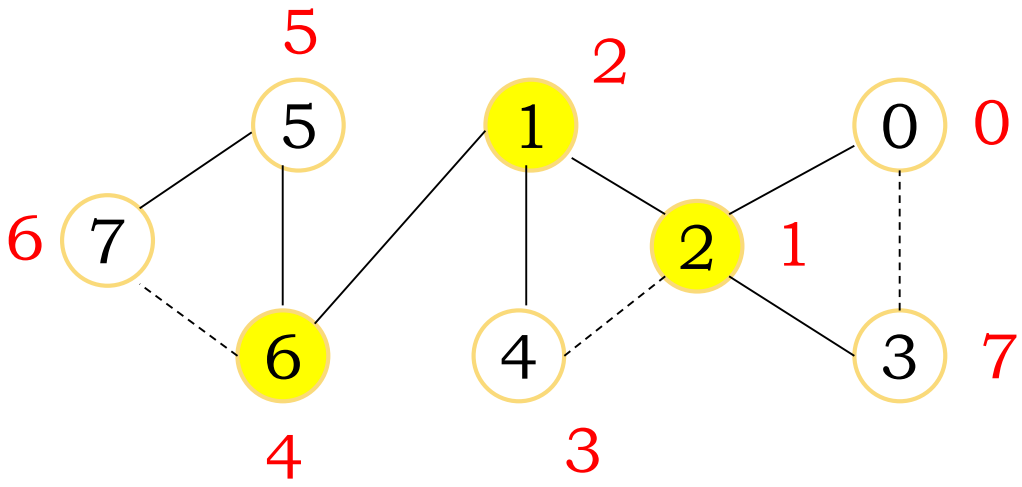


图 深度优先搜索和深度优先数

图中每个结点边上的偶对(d,Low)代表该结点的相应值。

例如，

结点2边上的(1,0)代表d[2]=1，Low[2]=0.

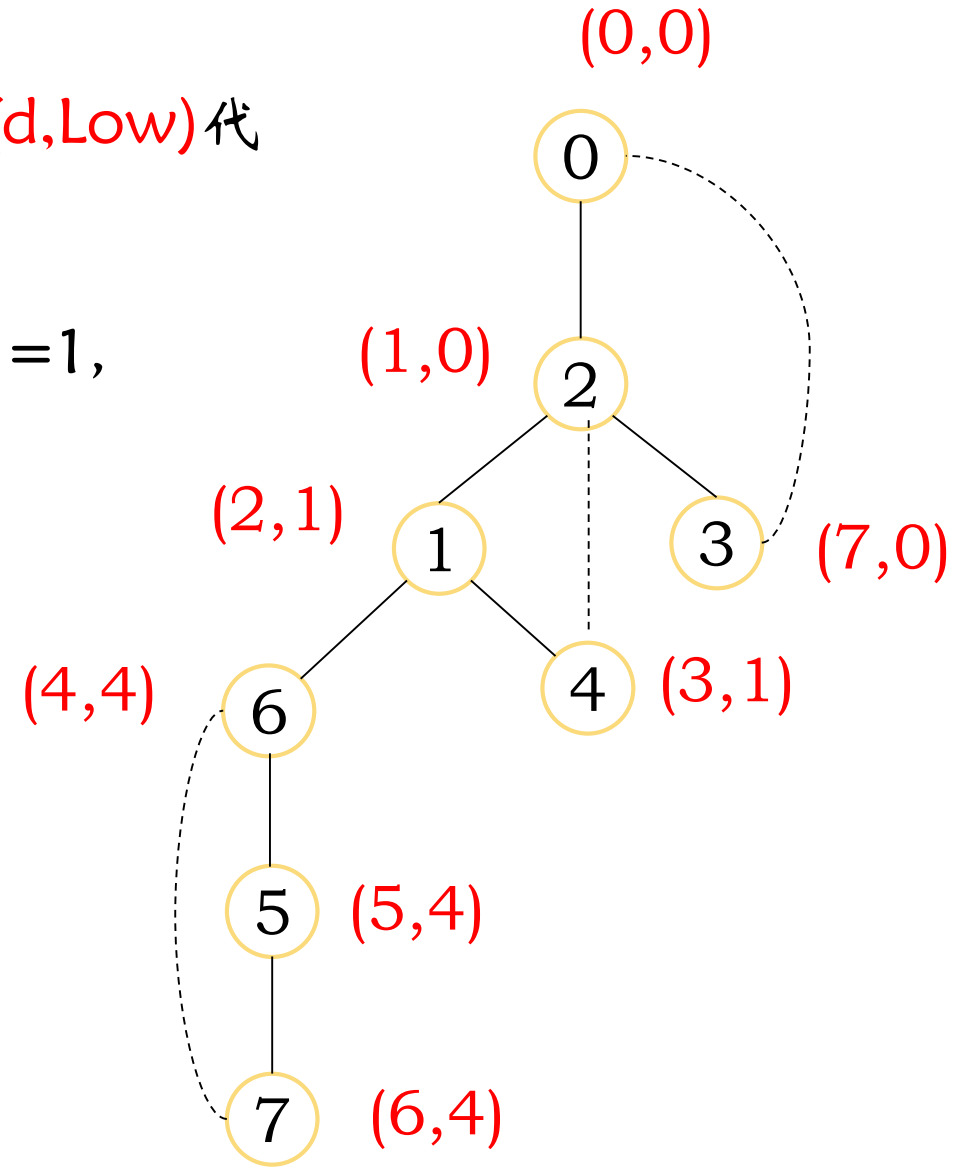


图 结点的d和Low值

构造双连通图

发现关节点和求双连通分量的目的是为了构造双连通网络，提高可靠性，改善网络的性能。

一个非双连通的图，如果已经求得原图的关结点和双连通分量，便可对原图添加若干边使之成为双连通图。

添加边算法的伪代码

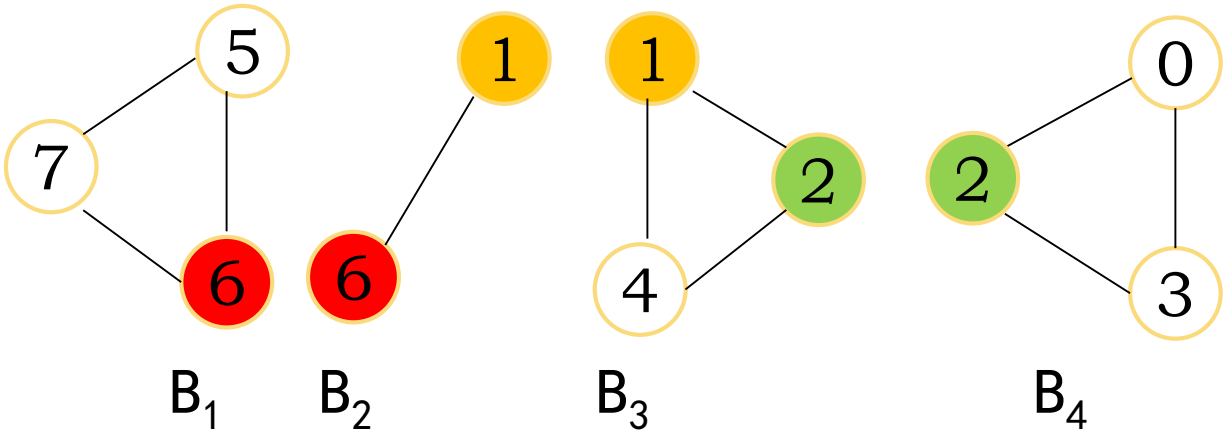
(1) for（图G的每个关节点a）{

(2)　　设$B_1, B_2, \cdots, B_k$为包含a的双连通分量；

(3)　　令v_i（v_i!=a）是B_i中的一个结点；

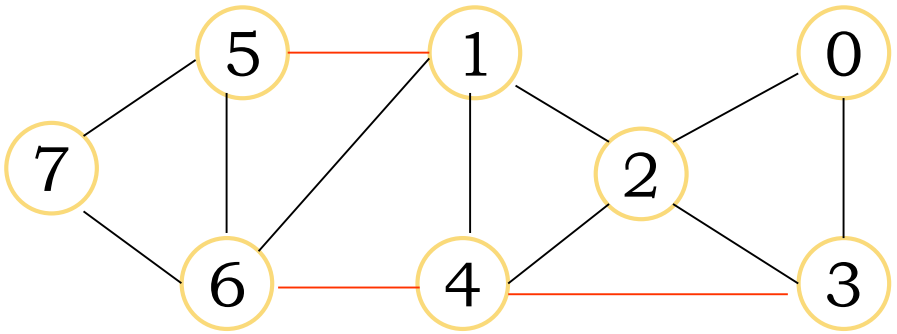
(4)　　将边（v_i, v_{i+1}），添加到图G中；

(5)　}



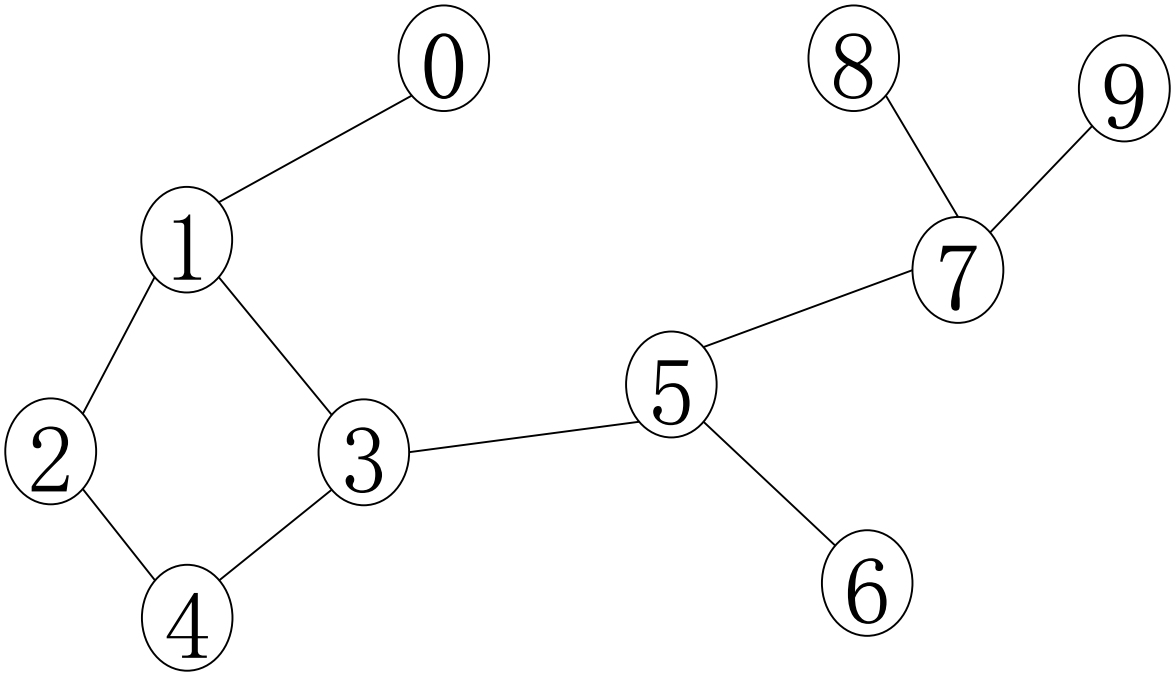
双连通分量

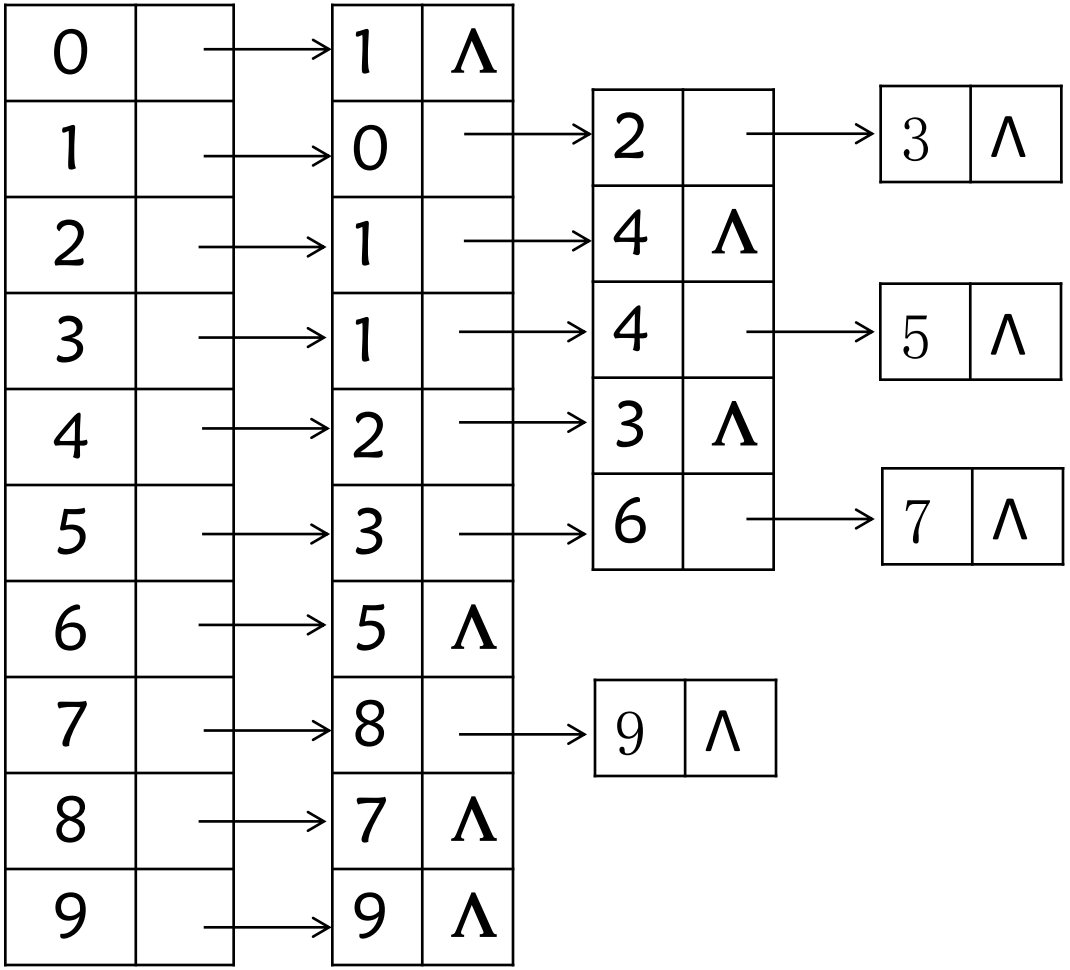
使用上述算法，只需添加3条边(5,1)、(6,4)和(4,3)，就可将原图改造成双连通图。



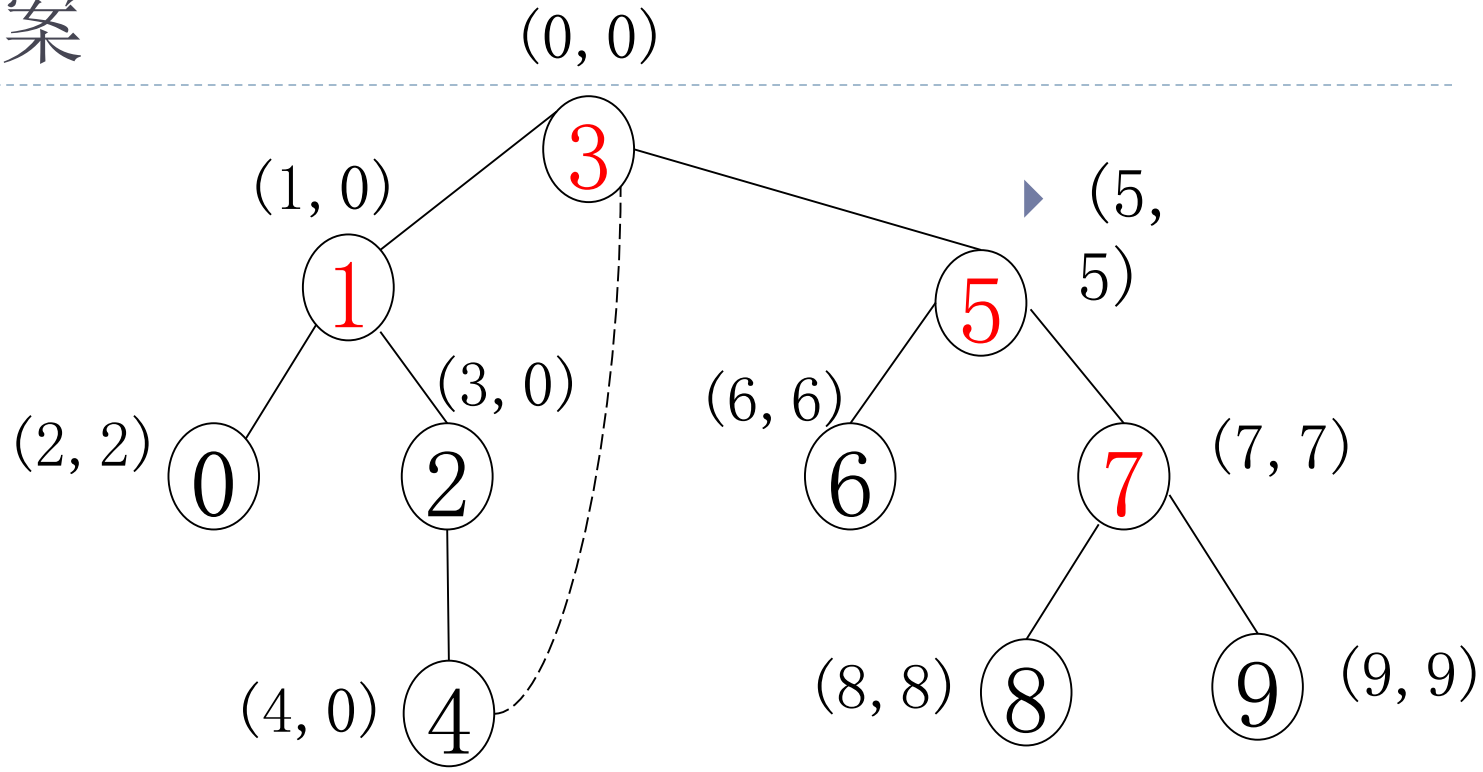
构造双连通图

- 对于下图G若从结点3开始深度优先搜索，画出图的深度优先生成树，计算各节点的d和low的值，并在图上标注（d,low）的值，确定所有关节点，然后画出该图的双连通分量（邻接表见下页）。



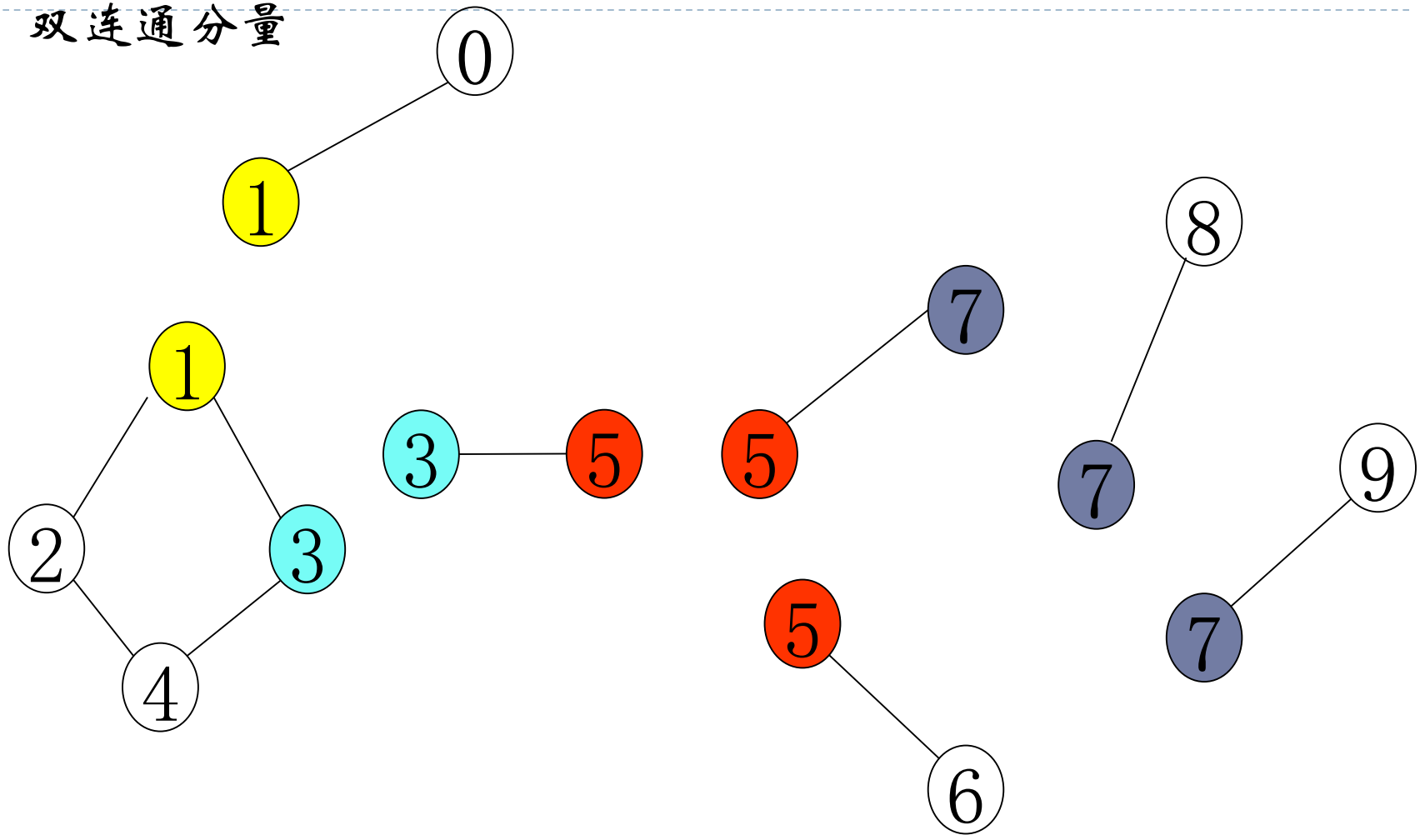

0
1
2
3
4
5
6
7
8
9
1 Λ
0
1
1
2
5 Λ
8
7 Λ
9 Λ
2
4 Λ
4
3 Λ
6
9 Λ
3 Λ
5 Λ
7 Λ

参考答案



关节点为1, 3, 5, 7

双连通分量



网络页面检索

- 深度优先搜索是一种在开发爬虫早期使用较多的方法。它的目的是要达到被搜索结构的叶结点(即那些不包含任何超链的HTML文件)。在一个HTML文件中，当一个超链被选择后，被链接的HTML文件将执行深度优先搜索，即在搜索其余的超链接结果之前必须先完整地搜索单独的一条链。深度优先搜索沿着HTML文件上的超链走到不能再深入为止，然后返回到某一个HTML文件，再继续选择该HTML文件中的其他超链。当不再有其他超链可选择时，说明搜索已经结束。
- 优点是能遍历一个Web 站点或深层嵌套的文档集合；缺点是因为Web结构相当深,，有可能造成一旦进去，再也出不来的情况发生。

广度优先搜索(Breadth-First Search, BFS)

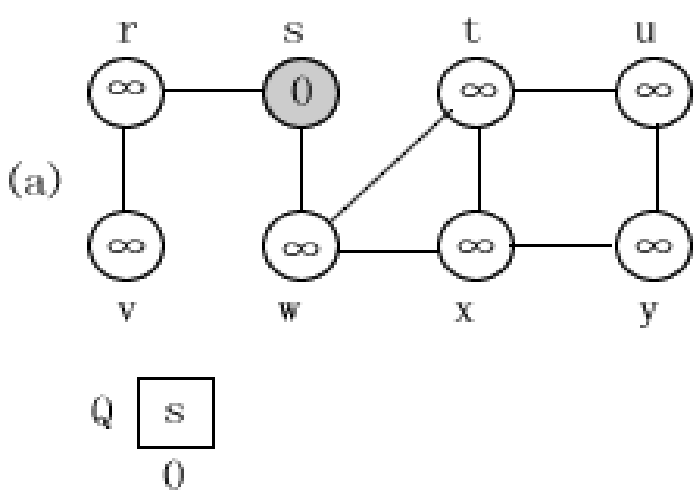
- 思路：在访问一个顶点v后，接下来依次访问邻接于v的所有顶点。对应的搜索树称之为广度优先搜索生成树。
- 实现：使用队列(Queue)
- BFS同样既适用于有向图，亦适用于无向图。
- 在无向图中，边分为：树边或者是横跨边。
- 在有向图中，边分为：树边，回边及横跨边。不存在前向边。

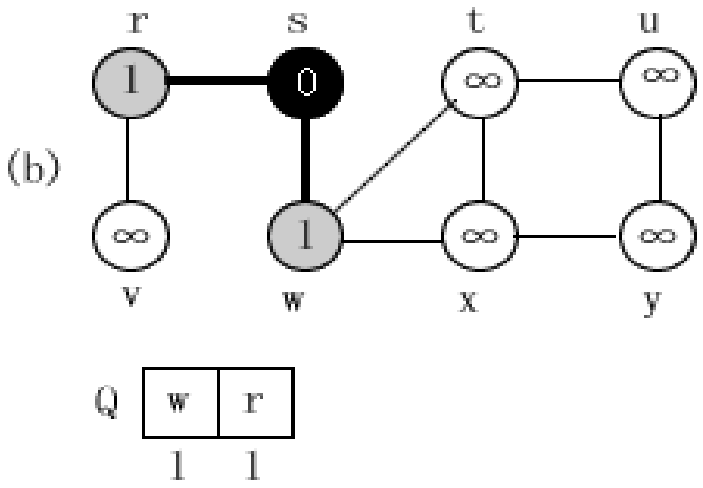
BFS算法
输入：无向图或有向图G= (V,E)
输出：广度优先搜索树中每个顶点的编号(编号：按广度优先的原则，该顶点被访问的次序)

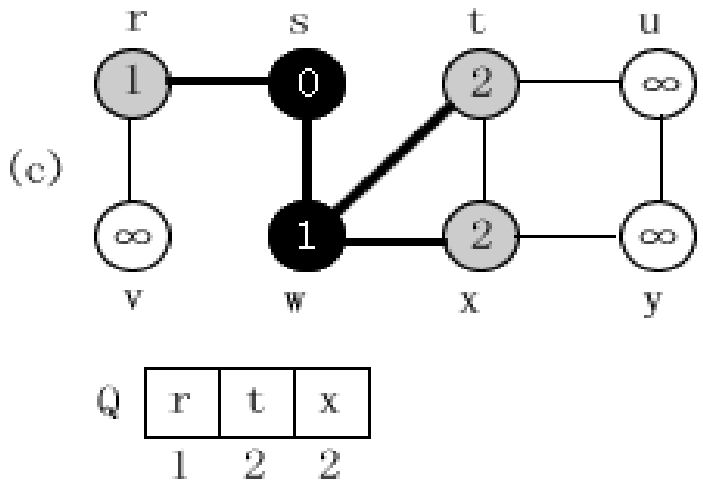
```
1. bfn←0
2. for v∈V
3.   visited[v] ←false
4. end for
5. for v∈V
6.    if visited[v] = false then bfs(v)
7. end for
```

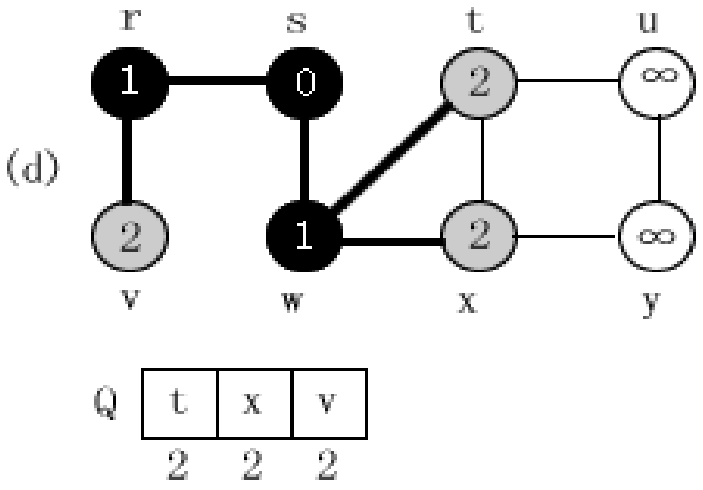
```
bfs(v)
1.  Q←{v}
2.  visited[v] ←true
3.  while Q≠{} //to be visited
4.     v←pop(Q)
5.     bfn←bfn+1
6.     for (v,w)∈E
7.        if visited[w]=false then
8.           Push(w,Q)
9.           visited[w]←true
10.       end if
11.    end for
12. end while
```

举例：广度优先搜索在一个无向图（结点S开始）上的执行过程

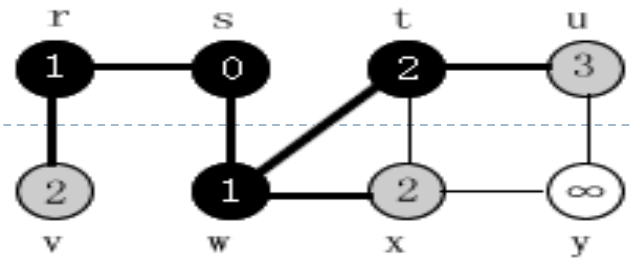






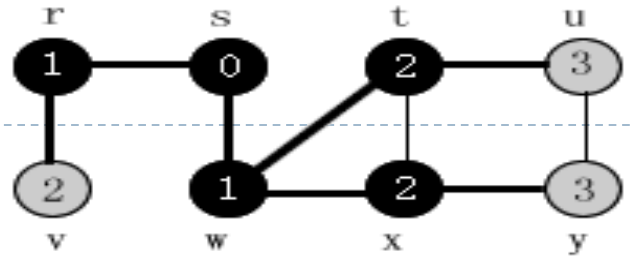


(e)



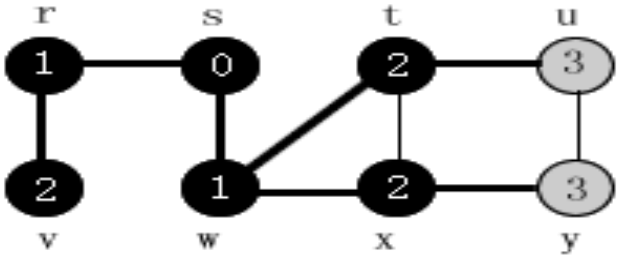
Q	x	v	u
	2	2	3

(f)



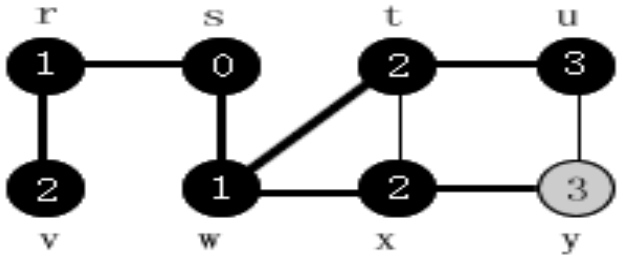
Q	v	u	y
	2	3	3

(g)



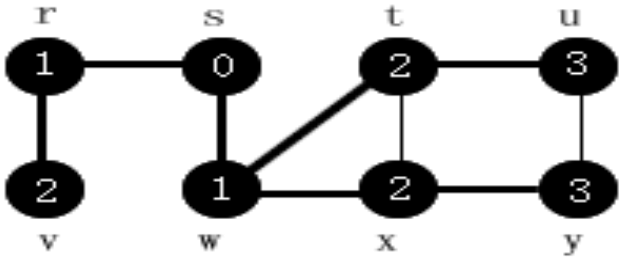
Q	u	y
	3	3

(h)



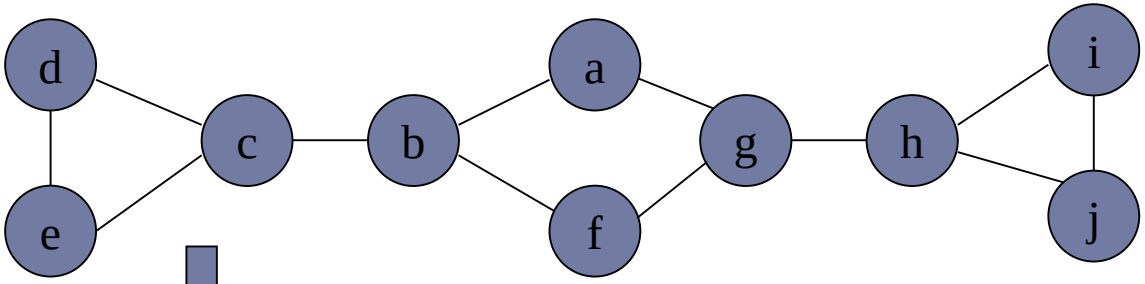
Q	y
	3

(h)



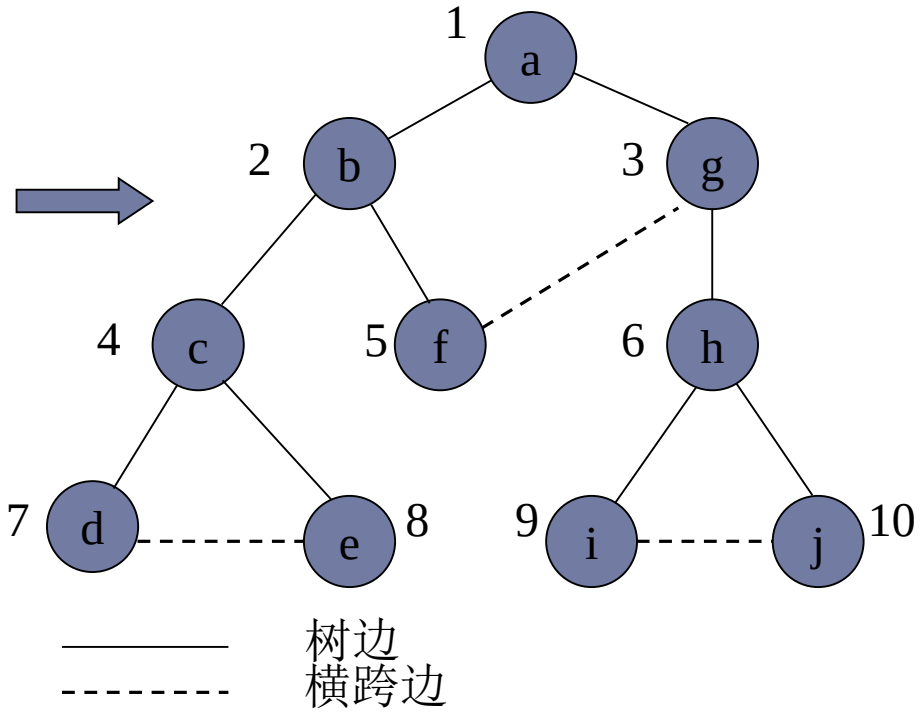
Q Φ

无向图BFS实例



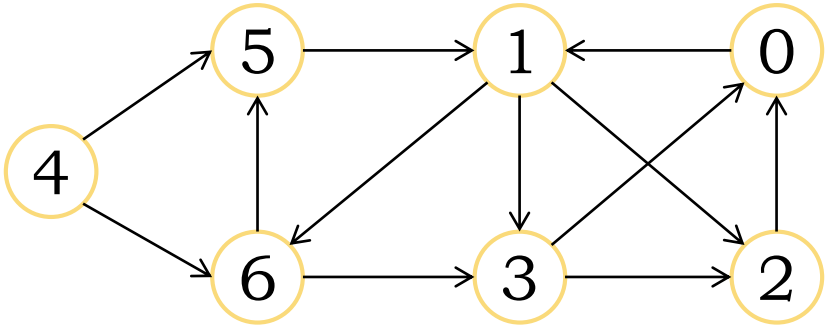
Queue

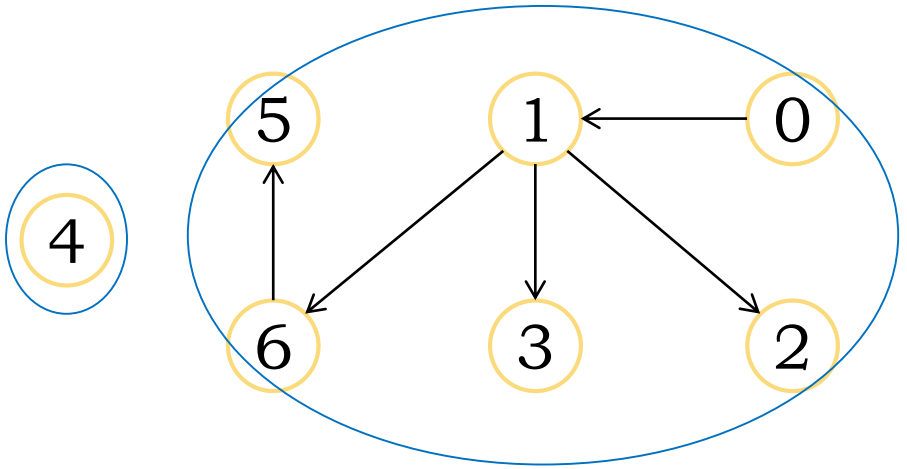
a	b	g	c	f	h	d	e	i	j



时间复杂度： $\Theta(m+n)$

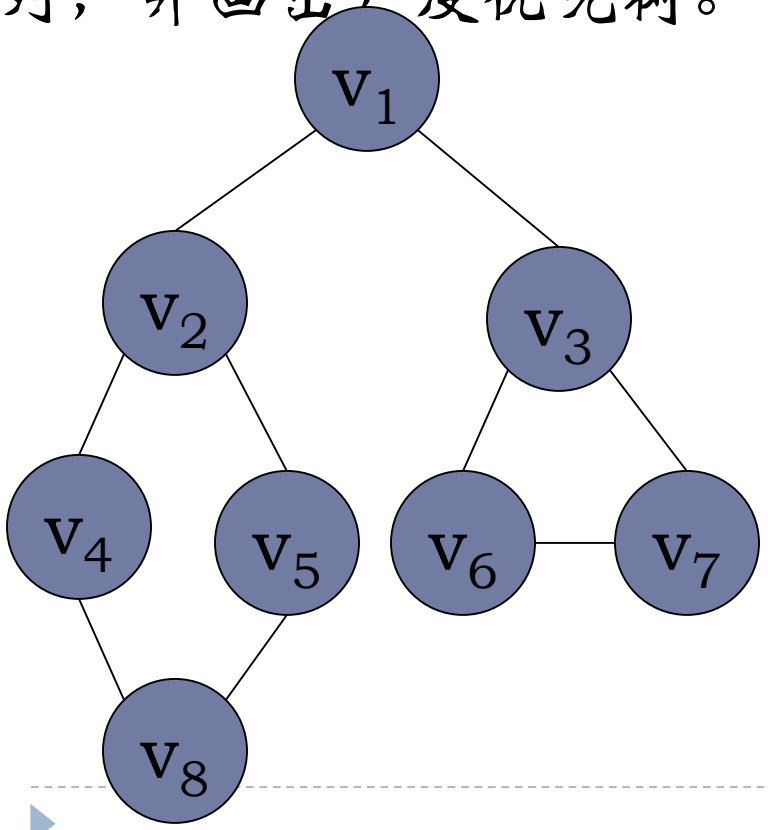
图中给出了以结点0为起点，广度优先遍历有向图G所得到的广度优先森林，它包含两棵广度优先树。

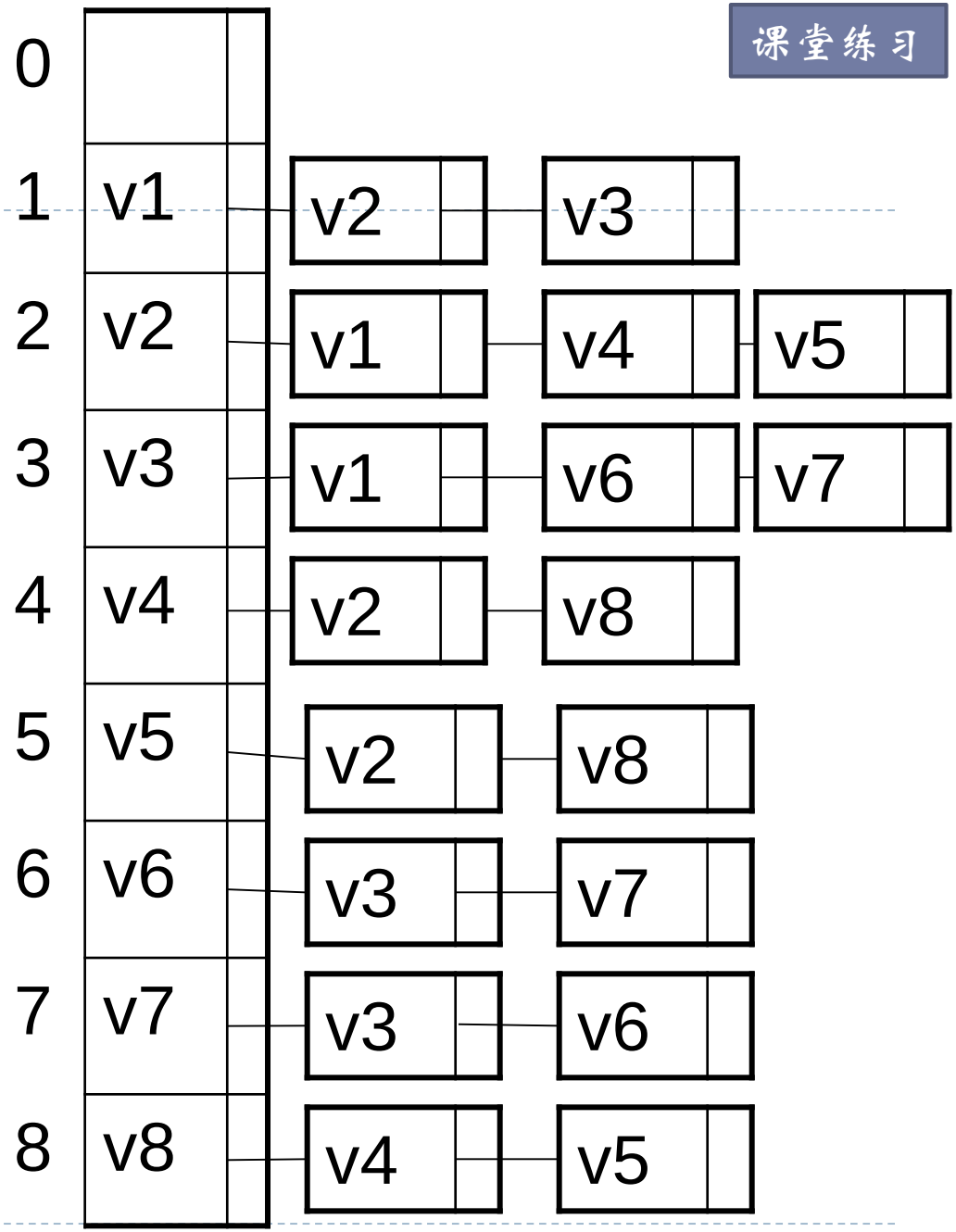




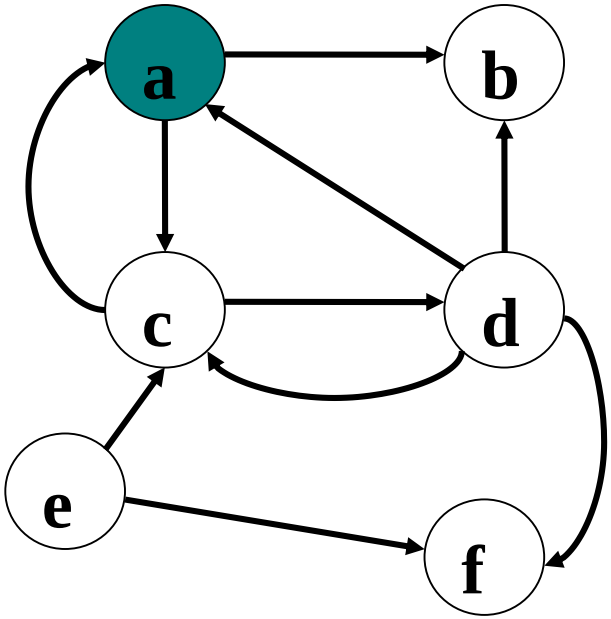
图G的广度优先森林

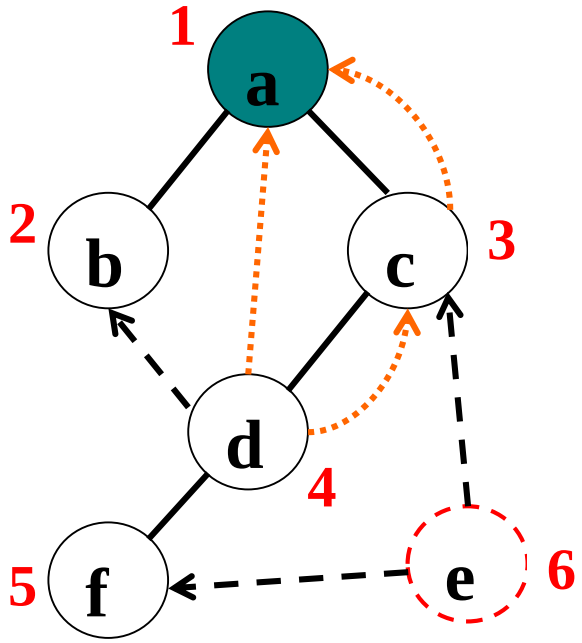
对于下面一个图及其存储结构，写出以v_2、v_8为起始点的广度优先遍历序列，并画出广度优先树。





有向图BFS实例





思考：为什么有向图的BFS中不会出现前向边。

1.前向边(Forward edges)—在迄今为止所构建的搜索生成树中，w是v的后裔，并且在探测(v,w)时，w已经被标记为"visited",则(v,w)为前向边。
2.既然要w是v的后裔，那么可以断定，w所在层较v所在的层要低；另一方面，广度优先搜索生成树是逐层产生的，即后裔顶点总是在祖先顶点之后访问。