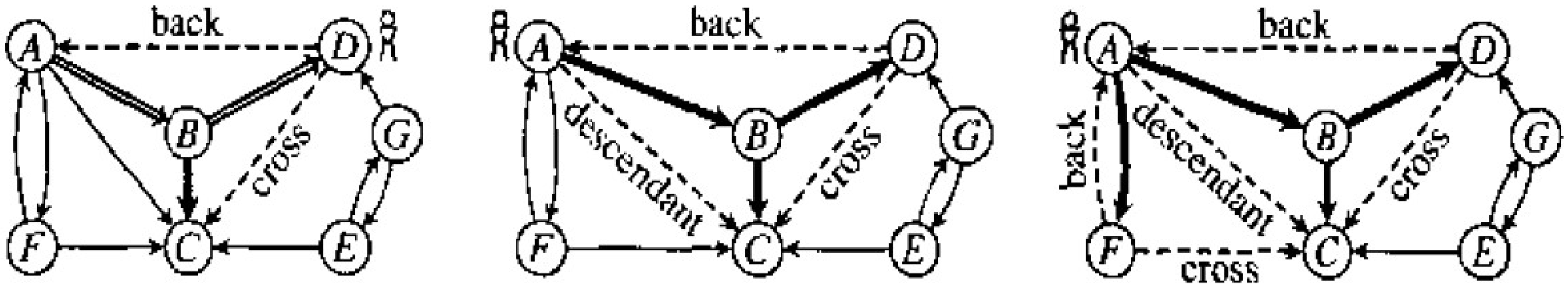
7.4 图搜索的应用--深度优先搜索生成树/森林

- 深度优先搜索生成树/森林
- 有向图边的“分类”，在深度优先搜索过程探查到边vw时：

1. 若w是undiscovered的，则vw称为tree edge，v变成w的父节点（深度优先搜索生成树）
2. 若w是v的祖先，则vw称为back edge（包括vv这种情况）
3. 若w是v的子孙，且在此前已经被discovered，则vw称为descendant edge
4. 若w和v没有任何ancestor/descendant关系，则vw称为cross edge

有向图边的“分类”



无向图的边只有：tree edge和back edge

深度优先搜索一个令人感兴趣的性质是通过搜索对图中的边进行归类，可以收集有关图的很多重要信息。例如：可以证明一个有向图无回路，当且仅当图的深度优先搜索没有产生回边

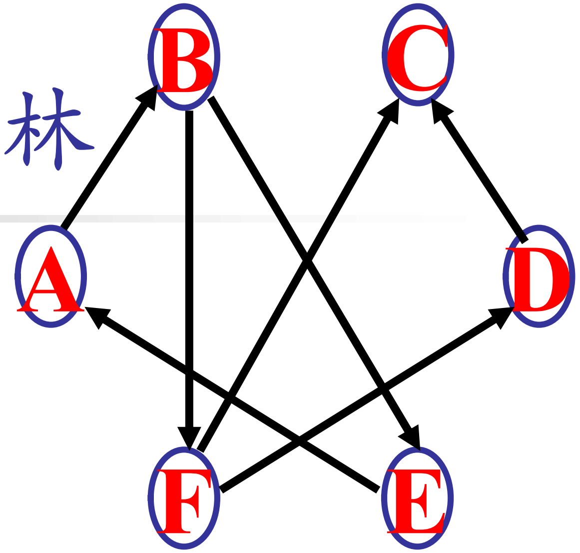
深度优先搜索生成树/森林

- 设计时器time，初始为0，每访问一个节点time+1，每结束一个结点的深度优先搜索time+1
- 深度优先搜索过程记录每个结点v的discoverTime(v)和finishTime(v)
- discoverTime(v)：结点v的被访问的时间
- finishTime(v)：从结点v出发做深度优先搜索结束的时间
- 利用时间可以判断一条边是否为回边等

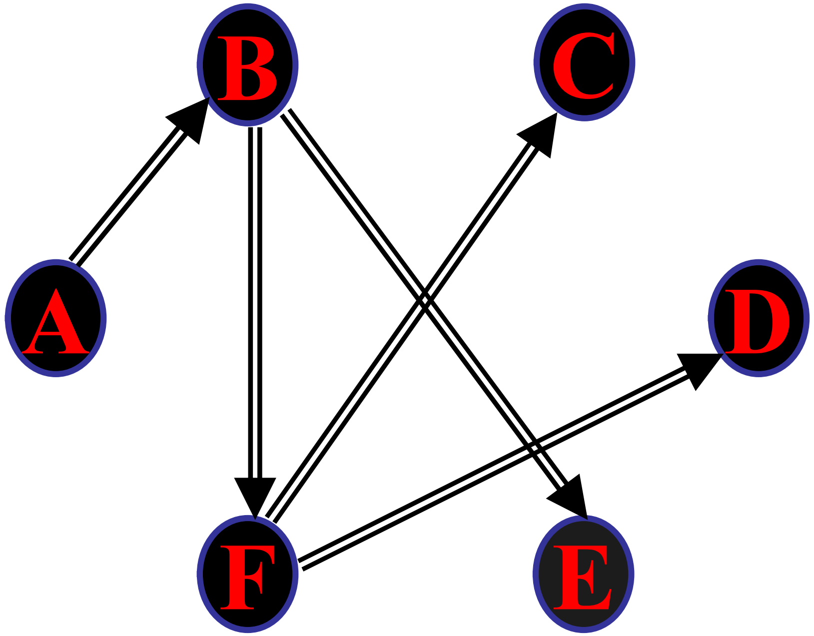
深度优先搜索生成树/森林

- active(v)=discoverTime(v).. finishTime(v)
- 证明：在图G的深度优先搜索森林中，u是v的子孙当且仅当discoverTime(v)<discoverTime(u)<finishTime(u)<finishTime(v)
  - 结点v的生存周期包含结点u的生存周期（从v出发做深度优先搜索时访问的u,所以u迟于v被访问，从u出发做深度优先搜索结束早于从v出发做深度优先搜索结束）---- u是v的子孙

深度优先搜索生成树/森林



- 根结点的确定
- 生成树（森林）的存储方式

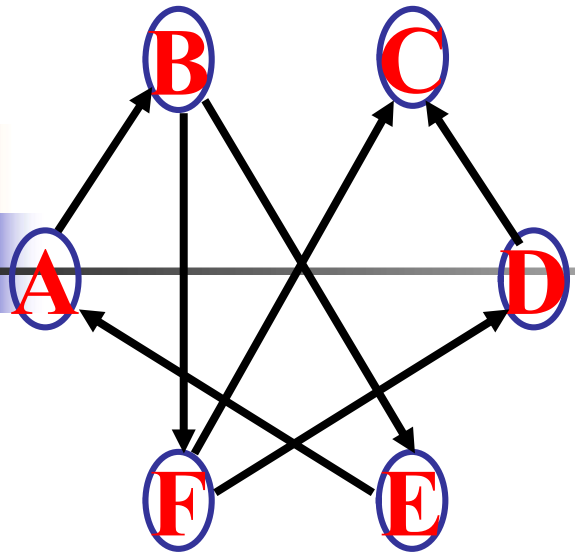


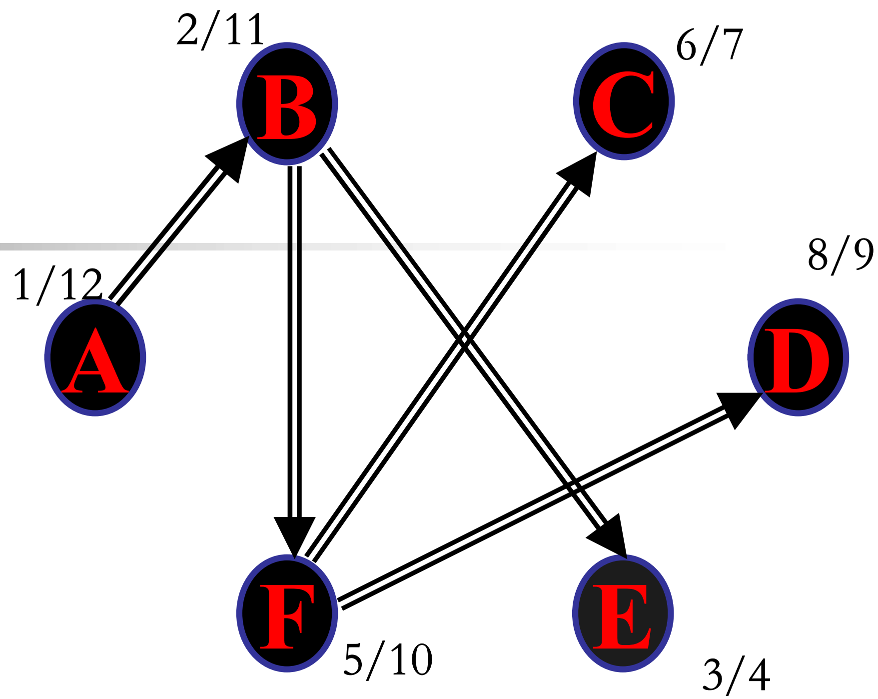
	data	parent
0	A	-1
1	B	0
2	C	5
3	D	5
4	E	1
5	F	1

深度优先搜索生成树/森林

```
void dfsSweep(G) {
        color all vertices to white;
         time=0;
        for each vertex v∈G
               if v is white
                        datas[v].parent=-1;
                        dfs(G, v);}
```

```
void dfs(G, v) {
    time++; discoverTime[v]=time; color v as gray;
    for each vertex w such that edge vw is in G
            if w is white
                        datas[w].parent=v;
                         dfs(G,w);
    time++; color v as black; finishTime[v]=time;
  }
```

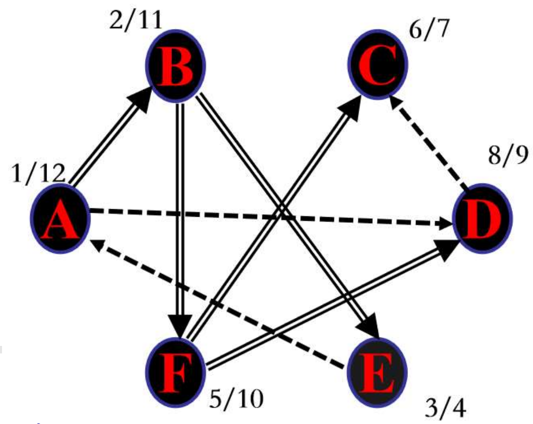




	data	parent
0	A	-1
1	B	0
2	C	5
3	D	5
4	E	1
5	F	1

证明：在图G的深度优先搜索森林中，u是v的子孙当且仅当 discoverTime(v)<discoverTime(u)<finishTime(u)<finishTime(v)

相关定理



Theorem 7.1: G是有向图，dfsSweep，对$v \in V$, $w \in V$

1. w is a descendant of v in the DFS forest if and only if $active(w) \subseteq active(v)$. If $w \neq v$ the inclusion is proper.
2. If v and w have no ancestor/descendant relationship in the DFS forest, then their *active* intervals are disjoint.
3. If edge $vw \in E$, then:
   a. vw is a cross edge if and only if $active(w)$ entirely precedes $active(v)$.
   b. vw is a descendant edge if and only if there is some third vertex x such that $active(w) \subset active(x) \subset active(v)$.
   c. vw is a tree edge if and only if $active(w) \subset active(v)$, and there is *no* third vertex x such that $active(w) \subset active(x) \subset active(v)$.
   d. vw is a back edge if and only if $active(v) \subset active(w)$.

相关定理

Corollary 7.2 The vertices that are discovered while v is active are exactly the descendants of v in its depth-first search tree. □

Theorem 7.3 (White Path Theorem) In any depth-first search of a graph G, a vertex w is a descendant of a vertex v in a depth-first search tree if and only if, at the time vertex v is discovered (just before coloring it gray), there is a path in G from v to w consisting entirely of white vertices.

定理7.3（白色路径定理）：在图G的深度优先搜索中，顶点w在深度优先搜索生成树中是顶点v的子孙节点当且仅当在顶点v被discover时，图G中存在一条从v到w的路径，路径上所有点均为白色

- 证明一个有向图G无回路，当且仅当图G的深度优先搜索没有产生回边(back edge)
- 证明:
  - →假设在图的深度优先搜索过程中产生回边(back edge)(u,v)，那么根据回边的定义，在深度优先搜索森林中顶点v是顶点u的祖先，则在图G中必存在从v到u的一条路径，该路径加上back edge(u,v)即构成了回路。
  - ←假设图G包含一回路C，且v是深度优先搜索过程中回路C上第一个被发现的顶点，在回路C存在边(v,w)和(u,v)。在discoverTime(v)时刻，C中的顶点形成了一条从v到u的白色路径（经过w），根据白色定理可知在深度深度优先搜索森林中顶点v是顶点u的祖先，……。

7.7 无向图的二连通分量

1. If one city's airport is closed by bad weather, can you still fly between every other pair of cities?

2. If one computer in a network goes down, can messages be sent between every other pair of computers in the network?

Problem 7.1

If any one vertex (and the edges incident upon it) are removed from a connected grapb, is the remaining subgraph still connected? ■

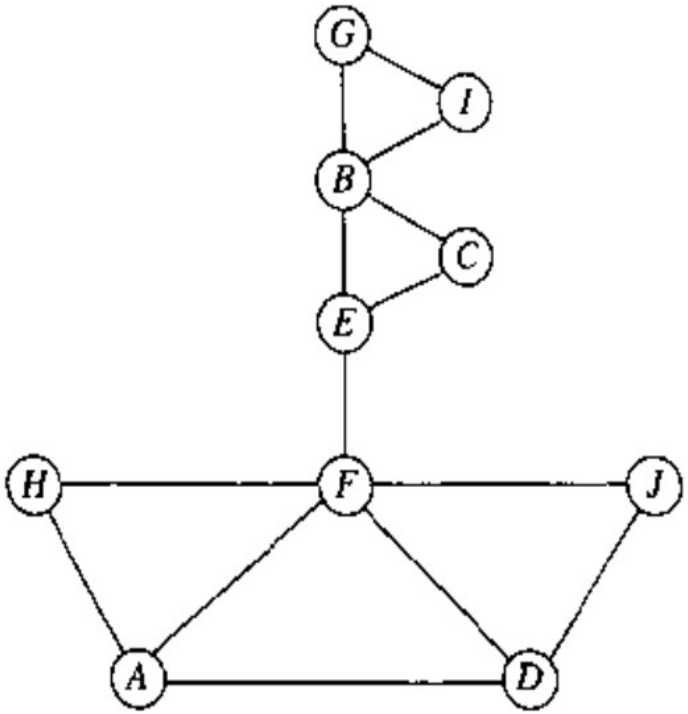
This question is important in graphs that represent all kinds of communication or transportation networks. It is also important to find those vertices, if any, whose removal can disconnect the graph. The purpose of this section is to present an efficient algorithm for answering these questions. This algorithm was discovered by R. E. Tarjan, and was one of the early algorithms that demonstrated the tremendous power of depth-first search.

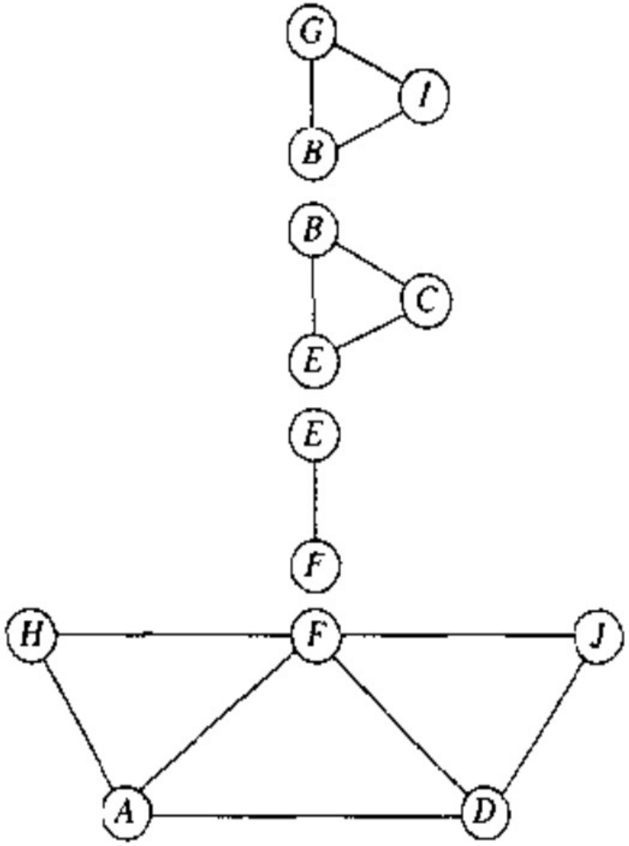
相关基本概念

- 一个连通图若删去任一顶点及其相关联的边，仍是连通图，则称其为**二连通图**。
- 二连通分量：无向图的一个**极大**二连通子图
- 割点：无向图**G**中若存在**2**个不同的点***x***，***w***，二者之间的任一路径都包含顶点***v***，则***v***为图**G**的一个割点
- 一个图是二连通的，当且仅当其不存在割点

二连通分量只是将边进行了划分，没有进行顶点划分

二连通分量图示





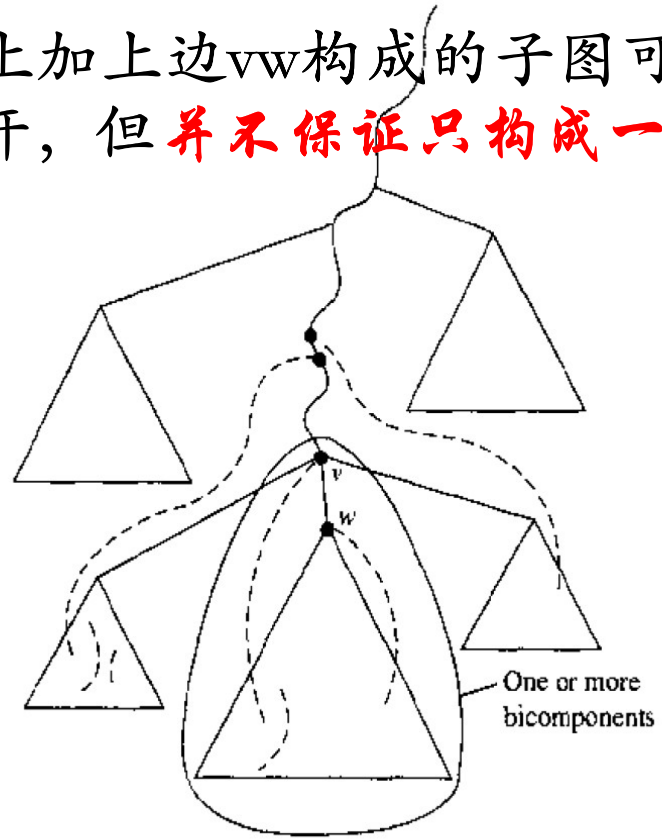
cut point割点

- 采用**深度**优先搜索对边进行划分，确定二连通分量
- 在采用**深度**优先搜索过程中，每一次从dfs返回到v时，判断v是否为割点

假设在深度优先搜索过程中，从dfs（G,w）返回到v，在以w为根的子树上的结点不存在回边指向v的真祖先，那么v是割点

cut point

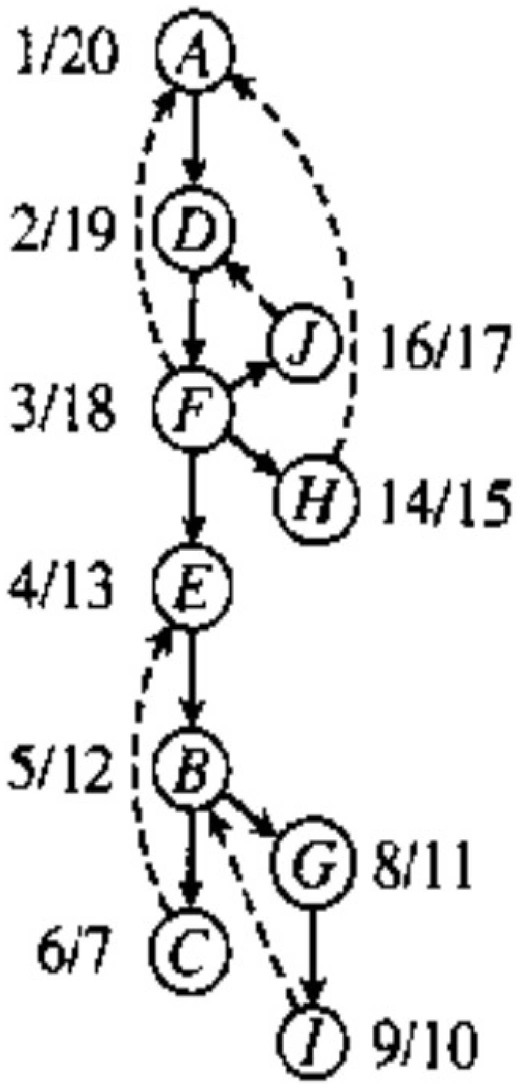
w为根的子树上加上边vw构成的子图可以和图的其余部分在v处分开，但**并不保证只构成一个二连通分量**



从深度优先搜索生成树最外层节点开始判断割点和**二连通分量**

二连通分量确定算法

- 顶点状态：***white***，***gray***，***black***
- 边：***tree edge***， ***back edge***
- ***v***若不是深度优先搜索树的**根**结点，搜索从***w***返回到***v***,若不存在回边从以***w***为根的子树到***v***的真祖先，那么***v***是一个割点。
- 跟踪一棵深度优先搜索生成树中的结点通过树边和**回边**最远到哪儿**----*back***
- 当**dfs**（**G,w**）结束时返回**---*back***
- **如何标识“远”**



- 用回边vw中结点w的时间discoverTime(w)标识回边vw回到多“远”

检测二连通分量的条件

- If v is first discovered, back=discoverTime[v];
- 当从v出发做深度优先搜索时，发现存在back edge(v,w)， back=min{back,discoverTime[w]}
- 当从w返回v时， 则 back=min{back, wBack}
- 当从w返回v时， 若wBack≥discoverTime[v]→发现一个二连通分量

```
void bicomponents(Graphs G,n)
    int v;
    IntStack edgeStack;
    for (v=0; v<G.vexnum; ++v)
    color[v] = white;
    time=0;
    edgeStack=creat();
    for(v=0; v<G.vexnum; ++v)
        if(color[v]==white)
            bicompDFS(G, color, v,-1)  ;
     return;
```

```
int bicompDFS(Graphs G,int [] color,int v,p)
    int w;  int back;
    time++; color[v]=gray; discoverTime[v]=time;
    back= discoverTime[v];
    for(pp=G.arcs[v].firstarc; pp!=NULL; pp=pp->link){
        w=pp->vex;
        if(color[w]==white)
            push(edgeStack,vw);
            int wBack= bicompDFS(G, color, w,v) ;
            if(wBack>=discoverTime[v])
                    initialize a new bicomponent;
                    pop and out the edgeStack down through vw.
            back=min{back,wBack}
        else if(color[w]==gray and w ≠p)
                push(edgeStack,vw);
                back=min{back, discoverTime[w]};
        }
    time++;finishTime[v]=time; color[v]=black;
   return back;
```

相关结论

Theorem 7.13 In a depth-first search tree, a vertex v, other than the root, is an articulation point if and only if v is not a leaf and some subtree of v has no back edge incident with a proper ancestor of v.

- 定理7.13 在深度优先搜索树中，若v不是根结点，那么v是割点当且仅当v不是叶子，v存在一棵子树，其上的结点不存在回边指向v的真祖先。

- 定理7.13 在深度优先搜索树中，若v不是根结点，那么v是割点当且仅当v不是叶子，v存在一棵子树，其上的结点不存在回边指向v的真祖先。

证明（->）在深度优先搜索树中，若v不是根结点且v是割点，那么一定存在2个结点x,y，那么x, y之间的任一条路径都经过v。那么x, y之间至少有一个是v的真子孙。所以v不是叶子。至少存在一个孩子（一棵子树）。

反证法：假设v的所有子树都存在回边指向v的真祖先

(1) x, y有一个是v的真子孙。

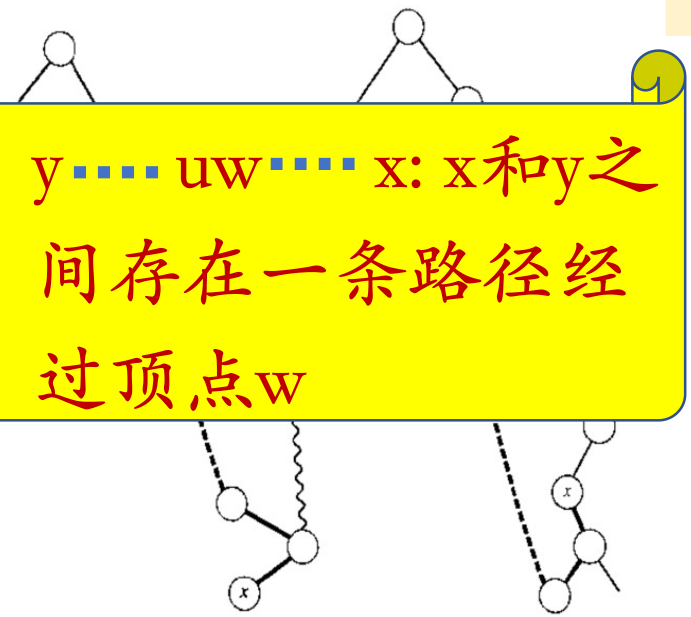
(2) x,y都是v的真子孙。一定位于v的2棵不同的子树

(1) x, y有一个是v的真子孙。不失一般性假设x是v的真子孙。x所在的v的那棵子树上存在顶点w，w产生一条回边wu指向v的真祖先u。

因为w和x在v的同一棵子树上，所以在这棵子树上w和x存在一条路径（不包含顶点v）。

x,y之间任一条路径都经过v，y和v有路径相通，那么:

（1.1）y也是v的真祖先，u和y都是v的真祖先，二者之间存在路径（不包含顶点v），这样x和y之间存在一条经过顶点w，而不经过顶点v的路径，与假设矛盾



y····uw····x: x和y之间存在一条路径经过顶点w

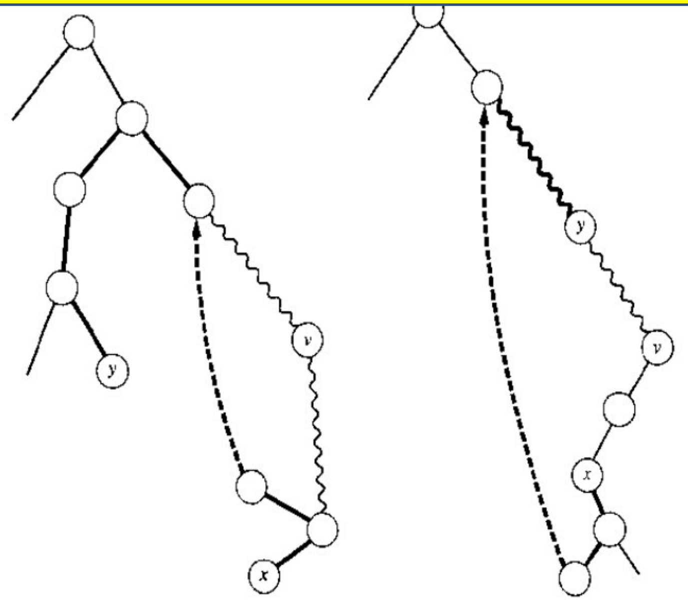
(1) x, y有一个是v的真子孙。不失一般性假设x是v的真子孙。x所在的v的那棵子树上存在顶点w，w产生一条回边wu指向v的真祖先u。

因为w和x在v的同一棵子树上，所以在这棵子树上w和x存在一条路径（不包含顶点v）。

x,y之间任一条路径都经过v，y和v有路径相通，那么:

（1.2） y不是v的真祖先，y不是v的真子孙，y和v有路径相通，则y和v位于某个顶点p的两棵不同的子树上，y和p有路径相通，p和u有路径相通， w和x存在一条路径（不包含顶点v），则这样x和y之间存在一条经过顶点w和p，，而不经过顶点v的路径假设矛盾。

y....p....uw....x: x和y之间存在一条路径经过顶点w



(2) x, y都是v的真子孙。一定位于v的2棵不同的子树。

x所在的v的那棵子树上存在顶点w，w产生一条回边wu指向v的真祖先u。y所在的v的那棵子树上存在顶点p，p产生一条回边pt指向v的真祖先t。

因为u和t都是v的真祖先，所以u和t存在一条路径（不包含顶点v）。

这样x和y之间存在一条经过顶点w，u，p和t而不经过顶点v的路径，与假设矛盾。

- 定理7.13 在深度优先搜索树中，若v不是根结点，那么v是割点当且仅当v不是叶子，v存在一棵子树，其上的结点不存在回边指向v的真祖先。

证明（<-）在深度优先搜索树中，若v不是根结点，v不是叶子，v存在一棵子树，其上的结点不存在回边指向v的真祖先，那么v是割点。

在深度优先搜索树中，因为v不是根结点，则v是存在父结点p。v不是叶子，假设v的一棵子树ST，其上的结点不存在回边指向v的真祖先，则ST上所有结点和p之间的路径都经过结点v，所以v是割点。