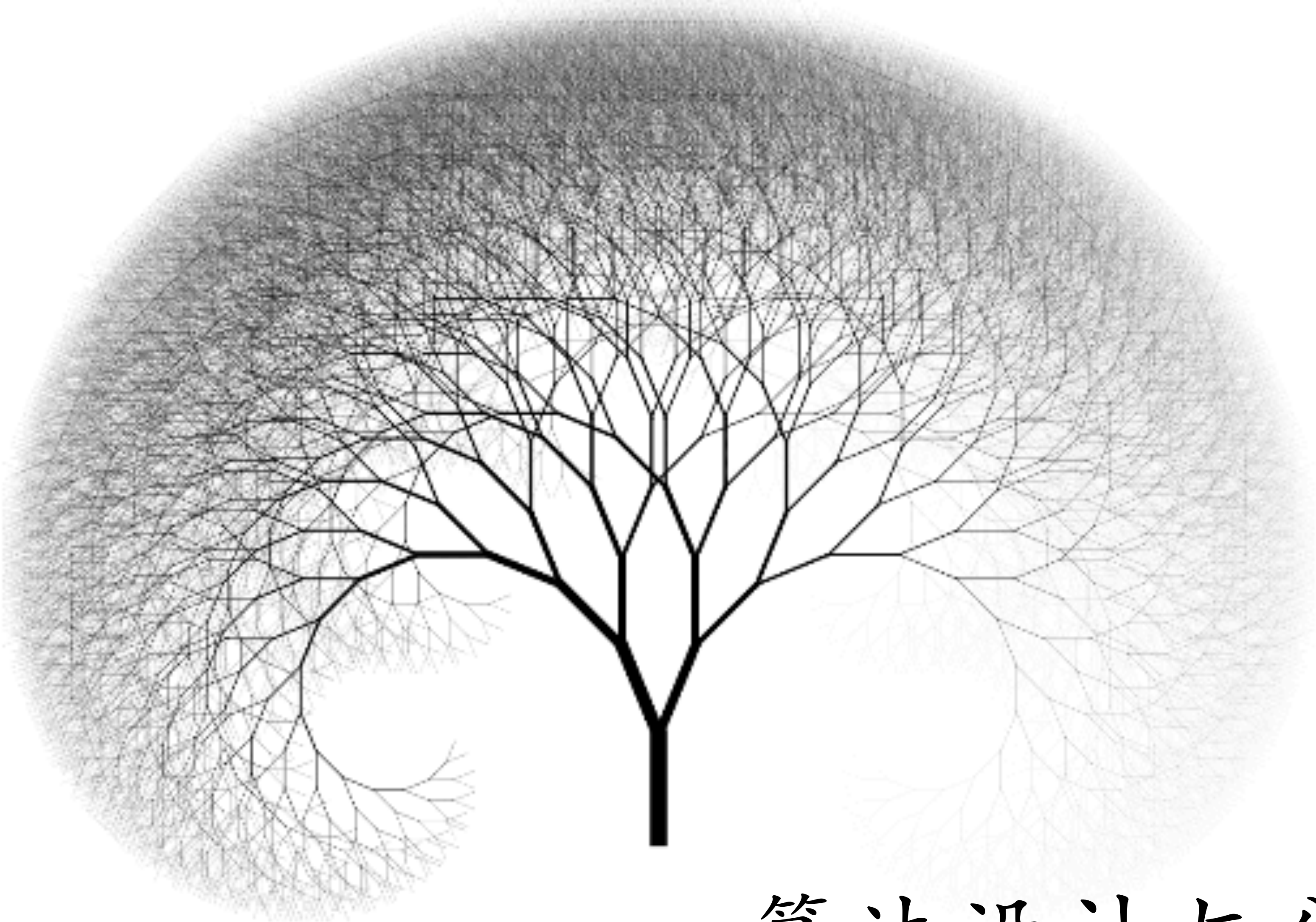


算法设计与分析

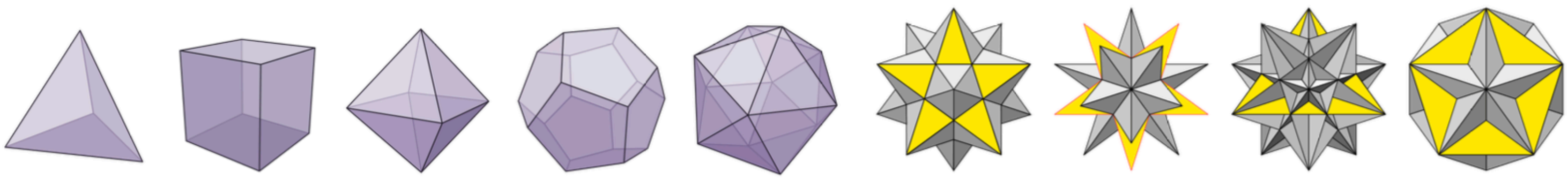
刘 安
苏州大学 计算机科学与技术学院
http://web.suda.edu.cn/anliu/

有趣的自然数

Interesting Natural Numbers

有趣的自然数

- 命题：任意一个自然数都是有趣的！
- 观测
  - 1：最小的自然数。有趣！
  - 2：唯一的偶素数。有趣！
  - 3：最小的奇素数。有趣！
  - 4：为地图着色所需的最大颜色数。有趣！
  - 5：不能用根式求解的一般多项式的最小次数。有趣！
  - 6：最小的完美数。有趣！
  - 7：不能用尺规作图完成的最小正多边形的边数。有趣！
  - 8：斐波那契数列中最后一个完全立方数。有趣！
  - 9：宇宙中正多面体的种数：5种凸正多面体+4种星形正多面体。有趣！



有趣的自然数

- 命题：任意一个自然数都是有趣的！
- 证明：
  - 假设不是所有的自然数都很有趣
  - 因此，必然有一个最小的不有趣的自然数n
  - 但是，最小的不有趣的自然数正是n的有趣之处
  - 所以，n既是有趣的又是不有趣的，矛盾！
  - 所以，任意一个自然数都是有趣的！

- 严格来说，该命题是不准确的，因为“有趣的”并无精确的定义
- 一般使用归纳法来证明某命题对所有的$n \in \mathbb{N}$都是真的
- 也可以使用反证法给出更加精妙的证明

反证

Contradiction

停机问题

- 给定一个程序P和一个输入i。一旦P在输入i上启动，它会终止吗？
- 定理：不存在程序$H(P(i))$，可以正确断定$P(i)$是否终止
- 证明
  - 假设存在这样的程序$H(P(i))$，如果$P(i)$可以终止，返回1，否则返回0
  - 构造程序$T(x)$，其中输入x是一个程序，具有自己的输入

```
void T(x):
1: if H(x) goto 1
```

  - 将程序T作为输入运行程序T，即$T(T)$
    - 如果T可以终止，那么$H(T)$返回1，从而进入无限循环，矛盾
    - 如果T不能终止，那么$H(T)$返回0，从而T可以终止，矛盾
  - 所以不存在这样的程序

深度优先搜索

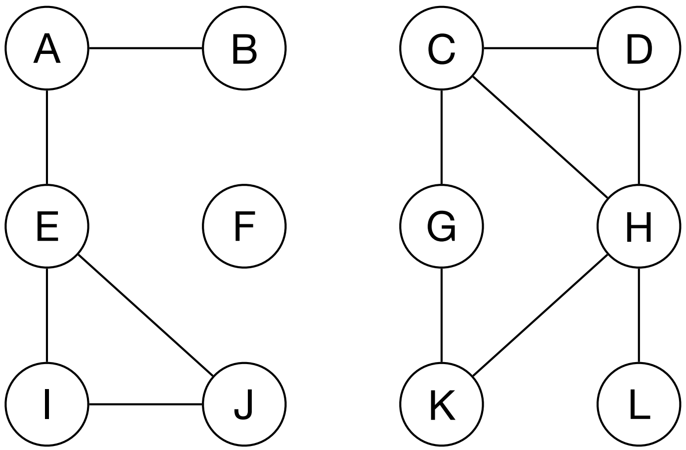
Depth-First Search

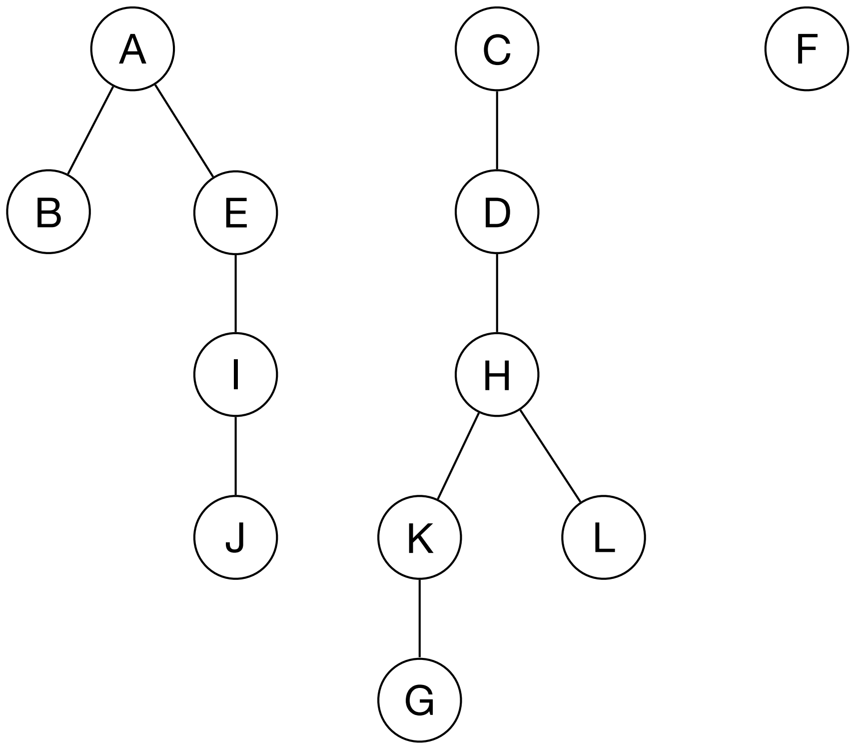
深度优先搜索

- 命题：explore过程结束后，所有从v可达的顶点都已被访问

procedure explore(G, v)
Input: $G(V, E)$ is a graph, $v \in V$
Output: visited(u) is set to true for all nodes u reachable from v

visited(v) = true
previsit(v)
for each edge $(v, u) \in E$:
    if not visited(u): explore(G, u)
postvisit(u)





深度优先搜索

- 命题：explore过程结束后，所有从v可达的顶点都已被访问

路径上的边数

- 证明（归纳法：对顶点到v的距离进行归纳）
  - 令u是从v可达的顶点，v, u之间的距离k是从v到u的最短路径的长度
  - 归纳假设：对于任意$k \geq 0$，与v间距不超过k的顶点都已被访问
  - 归纳步骤：考虑距离v等于$k+1$的顶点u
    - 令w是从v到u的最短路径上u的直接前驱，根据归纳假设，其被访问
    - 算法3-4行会访问w的所有邻居（必然包含u），所以命题成立

procedure explore(G, v)
Input: $G(V, E)$ is a graph, $v \in V$
Output: visited(u) is set to true for all nodes u reachable from v

1: visited(v) = true
2: previsit(v)
3: for each edge $(v, u) \in E$:
4:     if not visited(u): explore(G, u)
5: postvisit(u)

深度优先搜索

- 命题：explore过程结束后，所有从v可达的顶点都已被访问
- 证明（反证法）
  - 假设某个从v可达的顶点u在explore过程结束后未被访问
  - 考虑任意一条从v到u的路径
    - 令z是该路径中被访问的最后一个顶点，令其直接后继是w
    - 显然w未被访问
    - 然而根据算法3-4行，w作为z的邻居一定会被访问，矛盾！

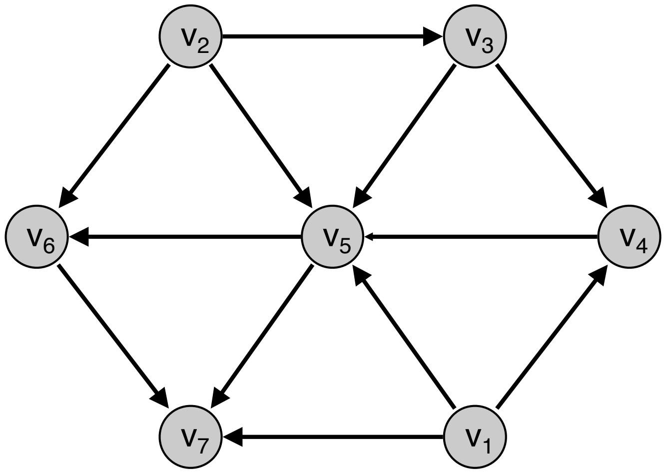
```
procedure  explore(G, v)
Input: G(V, E) is a graph, v ∈ V
Output: visited(u) is set to true for all nodes u reachable from v

1: visited(v) = true
2: previsit(v)
3: for each edge (v, u) ∈ E:
4:     if not visited(u): explore(G, u)
5: postvisit(u)
```

拓扑排序

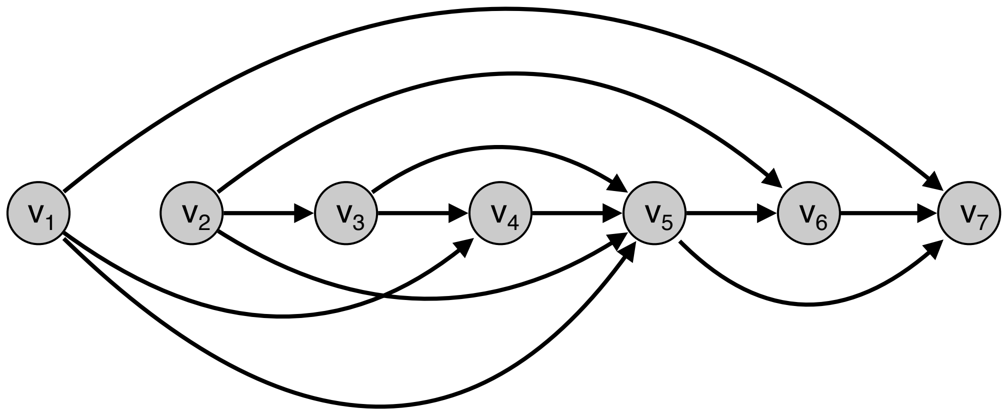
Topological Ordering

有向无环图和拓扑序

- 有向无环图：没有环的有向图
- 拓扑序：有向图G的拓扑序是指满足下列条件的顶点序列$v_1, v_2, \cdots, v_n$
  - 对于每一条边(v_i, v_j)，都有$i < j$



有向无环图G



G的一个拓扑序

有向无环图和拓扑序

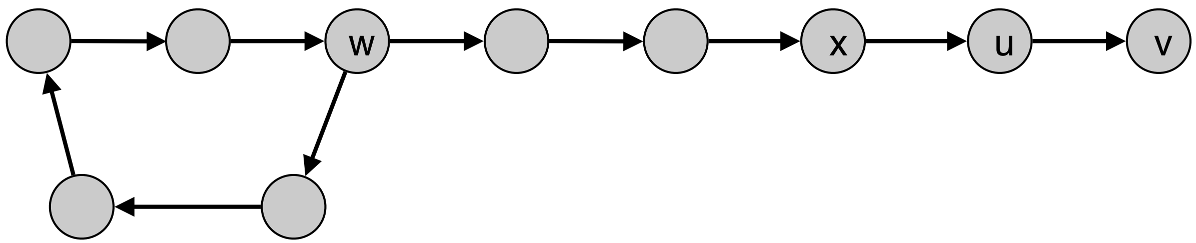
- 命题：如果有向图G有一个拓扑序，那么G是有向无环图
- 证明（反证法）
  - 假设G有一个拓扑序$v_1, v_2, \cdots, v_n$且G有一个环C
  - 令v_i是环C中具有最小下标的顶点，并且令v_j是v_i在环C中的直接前驱，显然(v_j, v_i)是一条边
  - 根据i的定义，有$i < j$
  - 然而，(v_j, v_i)是一条边且$v_1, v_2, \cdots, v_n$是一个拓扑序，所以$j < i$，矛盾！

有向无环图和拓扑序

- 命题：如果G是有向无环图，那么G中必然有一个源点 ← 没有边进入该点

汇点：没有边离开该点

- 证明（反证法）
  - 假设G是有向无环图且每一个顶点至少有一条进入的边
  - 任意选择顶点v，因为它至少有一条进入边(u,v)，所以可以沿着该边访问顶点u
  - 因为u至少有一条进入边(x,u)，所以可以沿着该边访问顶点x
  - 重复上述过程直至某个顶点w被访问两次（为什么w一定存在）
  - 令C是上述过程中访问的顶点序列，显然C是一个环，矛盾！



有向无环图和拓扑序

- 命题：如果G是有向无环图，那么G必然有一个拓扑序
- 证明（对顶点数量n进行归纳）
  - 基本情况：$n = 1$，显然成立
  - 归纳假设：G中顶点数小于n时命题成立
  - 归纳步骤：考虑具有n个顶点的有向无环图G
    - G必然有一个源点v
    - 将v及其关联的边从G中删除，得到的子图具有$n-1$个顶点，根据归纳假设，其有一个拓扑序$v_1, v_2, \cdots, v_{n-1}$
    - 因为没有边进入v，所以序列$v, v_1, v_2, \cdots, v_{n-1}$是$G$的一个拓扑序

基于递归的拓扑排序

- 命题：TopoSort_Rec可以在$O(m+n)$时间内找到图G的一个拓扑序
- 证明：
  - count(w)：当前进入w的边的数量，S：当前无进入边的顶点集合
  - 初始化上述两个变量的时间是$O(m+n)$：遍历所有的顶点及其边
  - 算法1-2行
    - 从集合S中删除一个元素v
    - 对于每一条边(v,w)，更新count(w)，如果其为0，将w加入集合S中
    - 每一条边关联操作的代价都是$O(1)$

procedure TopoSort_Rec(G)
Input: $G(V,E)$ is a directed acyclic graph
Output: A topological order of G

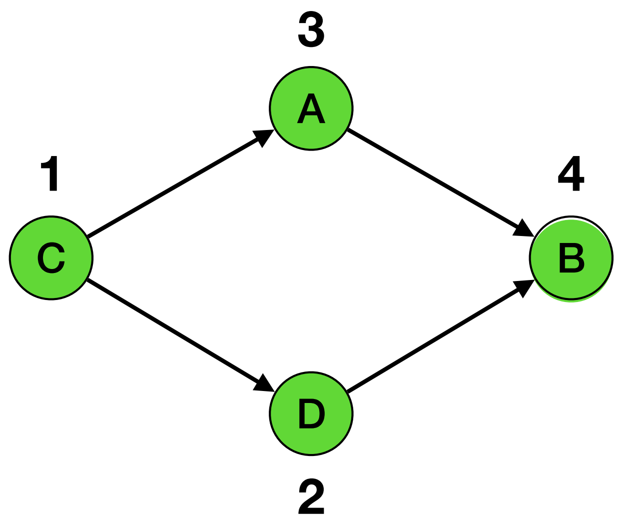
1: Find a node v with no incoming edges and order it first
2: Delete v from G
3: Recursively compute a topological order of $G-\{v\}$ and append this order after v

基于深搜的拓扑排序

- 命题：TopoSort的时间复杂度是$O(m+n)$。其结束运行后，每个顶点v都有一个f值，且这些f值构成了G的一个拓扑序

```
procedure  TopoSort(G)
Input: G(V, E) is a directed acyclic graph
Output: The f-values of vertices constitute a topological sorting of G

mark all vertices as unexplored
curLabel ← |V|
for every v ∈ V:
    if not visited(v): explore(G, v)
```

```
procedure  explore(G, v)
Input: G(V, E) is a graph, v ∈ V
Output: visited(u) is set to true for all nodes u reachable from v

visited(v) = true
for each edge (v, u) ∈ E:
    if not visited(u): explore(G, u)
f(v) ← curLabel
curLabel ← curLabel − 1
```



基于深搜的拓扑排序

- 命题：TopoSort的时间复杂度是$O(m+n)$。其结束运行后，每个顶点v都有一个f值，且这些f值构成了G的一个拓扑序
- 证明（时间复杂度省略）
  - 对于每一个顶点v，explore恰好被调用一次，且被赋予一个唯一的f值
  - 对于任意边(v, w)，下面证明必有$f(v) < f(w)$
    - v在w之前被访问
      - explore(G, w)在explore(G, v)执行过程中被调用
      - explore(G, w)先结束，所以$f(v) < f(w)$
    - w在v之前被访问
      - 不存在从w到v的路径
      - v不可能在explore(G, w)过程中被访问
      - explore(G, w)在explore(G, v)开始前结束，所以$f(v) < f(w)$

Quiz

What happens when the TopoSort algorithm is run on a graph with a directed cycle?

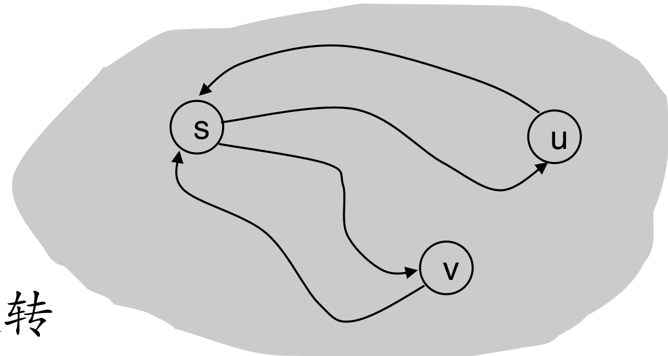
- **A.** The algorithm might or might not loop forever
- **B.** The algorithm always loops forever
- **C.** The algorithm always halts, and may or may not successfully compute a topological ordering.
- **D.** The algorithm always halts, and never successfully computes a topological ordering.

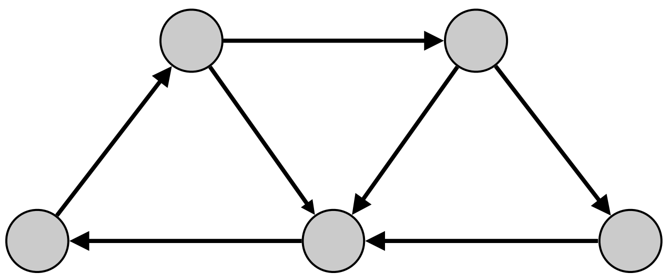
强连通分量

Strong Connected Components

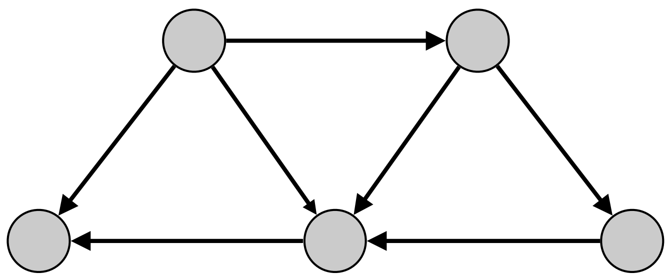
有向图的强连通性

- 有向图的强连通性：任意两个顶点之间是互相可达的
- 可以在$O(m+n)$时间内判定图G是否是强连通的
  - 随机选择一个顶点s
  - 在图G中从顶点s开始深度优先搜索
  - 构造图G的反转图G^r：将图G中所有边的方向反转
  - 在图G^r中从顶点s开始深度优先搜索
  - 如果所有顶点在两次深度优先搜索中均被访问，那么图G是强连通的





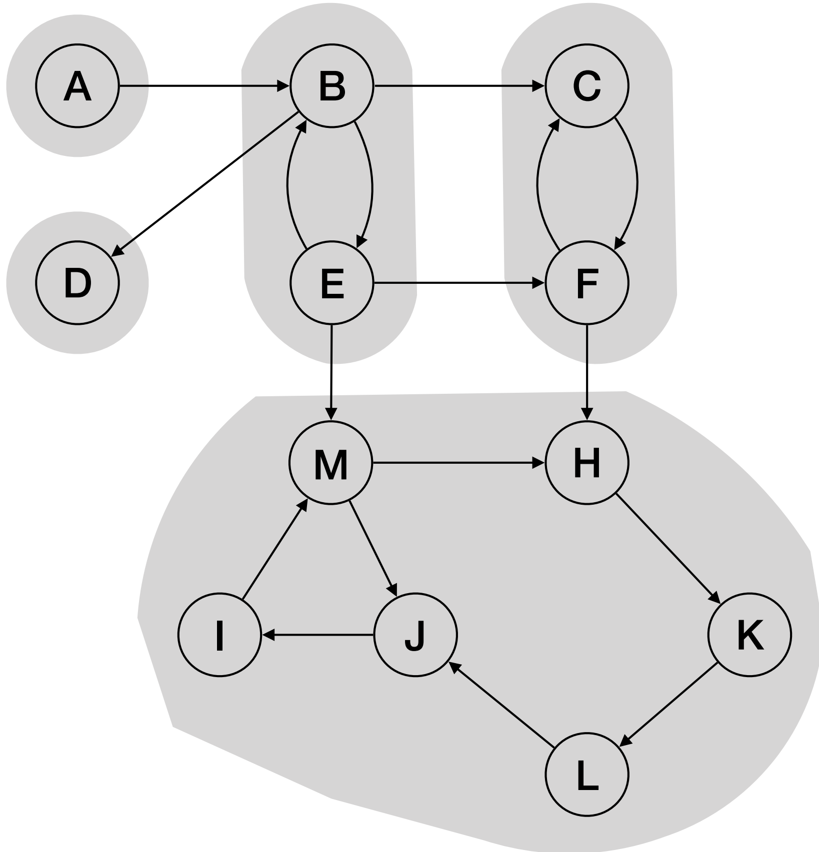
强连通



非强连通

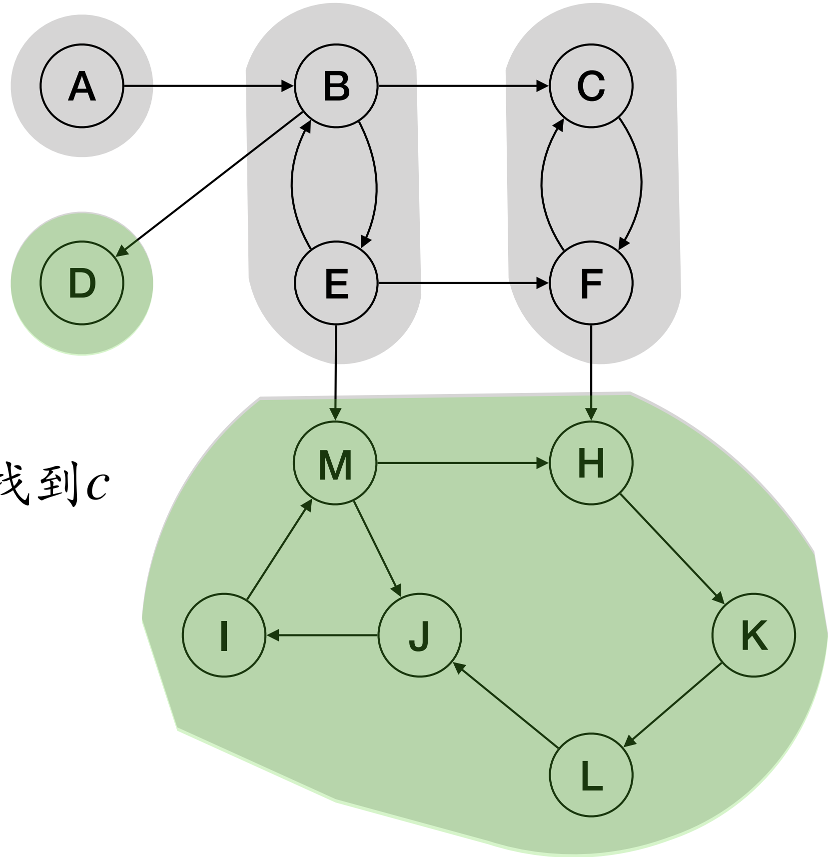
强连通分量

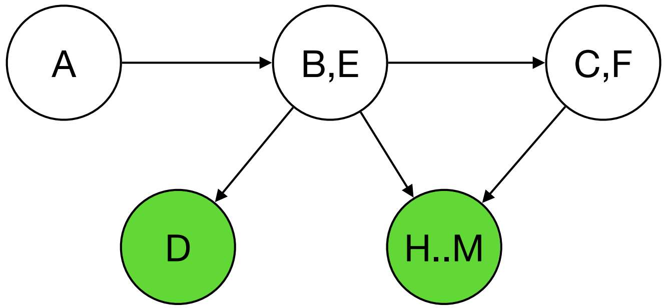
- 强连通分量：互相可达顶点的最大子集
- 问题：如何找到图G所有的强连通分量
- 观察
  - explore(G, A)：不能找到一个强连通分量
  - explore(G, D)：可以
  - explore(G, C)：不能
  - explore(G, K)：可以
- 结论：从哪个顶点开始搜索至关重要



强连通分量

- 强连通分量：互相可达顶点的最大子集
- 问题：如何找到图G所有的强连通分量
- 结论：从哪个顶点s开始搜索至关重要
- 考虑强连通分量组成的有向无环图
  - 如果s在汇点强连通分量c中，从s搜索可以找到c
- 如何找到一个必定在汇点强连通分量中的点
  - TopoSort给出的f值是否有帮助
    - 具有最大f值的顶点？具有最小f值的顶点？
  - 具有最小f值的顶点一定在源点强连通分量中
- 反转图G中的所有边，构造图G^r
  - 图G^r中具有最小f值的顶点即为所求





强连通分量的拓扑序

- 命题：令S_1, S_2是有向图G的两个强连通分量，且G中存在一条边(v, w)，其中$v \in S_1, w \in S_2$，那么$\min_{x \in S_1} f(x) < \min_{y \in S_2} f(y)$

- 证明
  - 考虑S_1, S_2中顶点的explore顺序，下面两种情况必居其一
    - 存在一个顶点$s \in S_1$，其在S_2中所有顶点被访问之前被访问
    - 存在一个顶点$s \in S_2$，其在S_1中所有顶点被访问之前被访问

强连通分量的拓扑序

- 命题：令S_1, S_2是有向图G的两个强连通分量，且G中存在一条边(v, w)，其中$v \in S_1, w \in S_2$，那么$\min_{x \in S_1} f(x) < \min_{y \in S_2} f(y)$
- 证明
  - 考虑S_1, S_2中顶点的explore顺序，下面两种情况必居其一
    - 存在一个顶点$s \in S_1$，其在S_2中所有顶点被访问之前被访问
    - 存在一个顶点$s \in S_2$，其在S_1中所有顶点被访问之前被访问
      - 任意顶点$t \in S_2$，存在从s到t的路径
        - S_1, S_2是强连通分量且存在边(v, w)
      - explore(G, t)在explore(G, s)执行过程中被调用
      - explore(G, t)先结束，所以$f(s) < f(t)$

强连通分量的拓扑序

- 命题：令S_1, S_2是有向图G的两个强连通分量，且G中存在一条边(v, w)，其中$v \in S_1, w \in S_2$，那么$\min_{x \in S_1} f(x) < \min_{y \in S_2} f(y)$

- 证明
  - 考虑S_1, S_2中顶点的explore顺序，下面两种情况必居其一
    - 存在一个顶点$s \in S_1$，其在S_2中所有顶点被访问之前被访问
    - 存在一个顶点$s \in S_2$，其在S_1中所有顶点被访问之前被访问
      - 任意顶点$t \in S_1$，不存在从s到t的路径
      - explore(G, s)执行过程中，S_2中的所有顶点将被访问，但S_1中的所有顶点都不会被访问
      - 对于所有的$t \in S_1, s \in S_2$，explore(G, s)在explore(G, t)开始前结束
      - 所以，$f(t) < f(s)$

Kasaraju算法

```
procedure Kasaraju(G)                                      时间复杂度: O(m + n)
Input: G(V, E) is a directed graph
Output: For every v, w ∈ V, scc(v) = scc(w) if and only if v, w are in the same SCC

1. G^r ← G with all edges reversed
2. TopoSort( G^r ) // compute f-values
3. mark all vertices of G as unexplored
4. numSCC ← 0
5. for each v ∈ V, in increasing order of f(v) // try all vertices in a magic order
6.    if not visited(v) // avoid redundancy
7.        numSCC ← numSCC + 1 // new component
8.        explore(G, v)
```

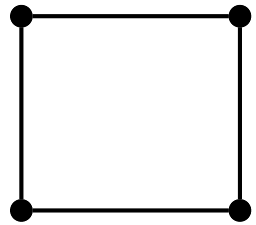
```
procedure explore(G, v)
1: visited(v) = true
2: scc(v) ← numSCC
3: for each edge (v, u) ∈ E:
4:    if not visited(u): explore(G, u)
```

双连通分量

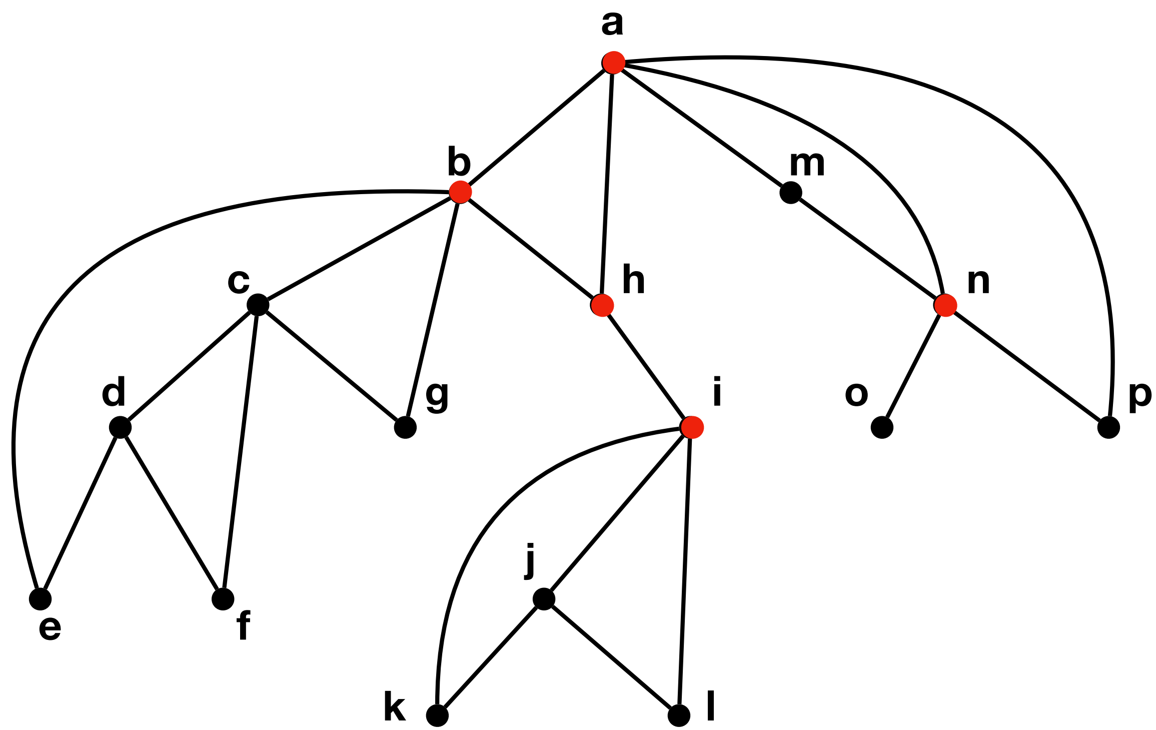
Biconnected Components

双连通图和双连通分量

- 双连通图：删除任一顶点（及其相关联的边）后依然连通的无向图
  - 任意两个顶点之间存在至少两条顶点不相交的路径

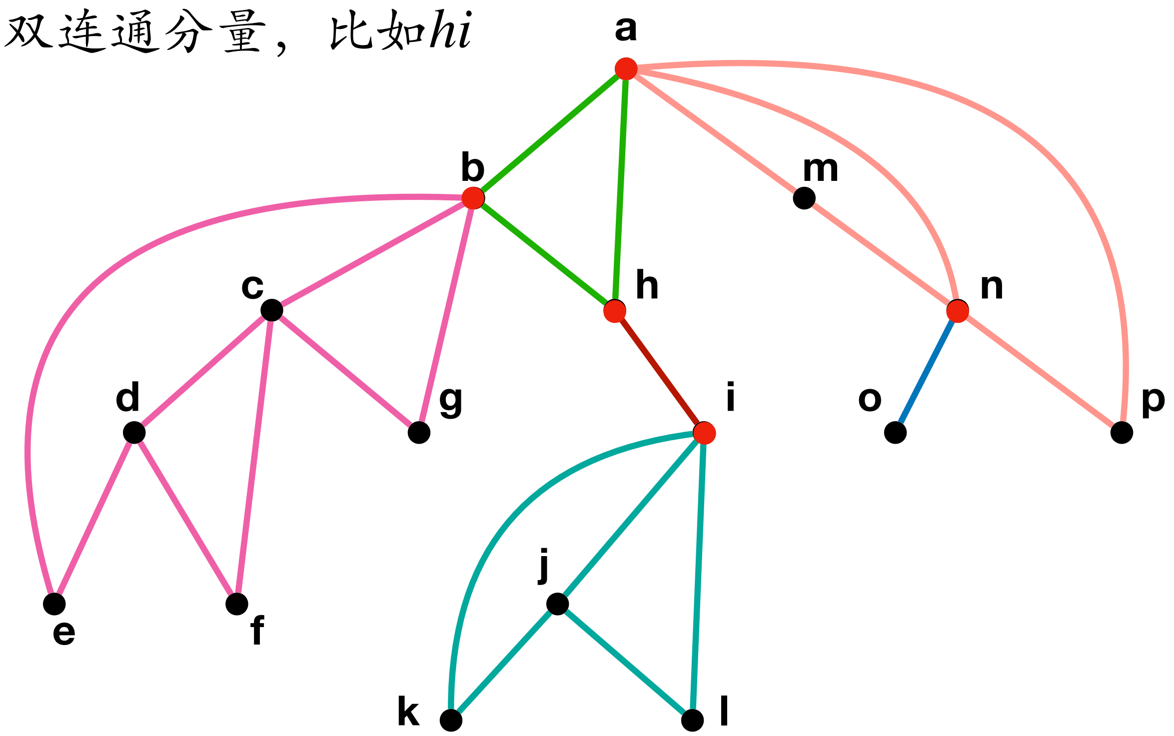


- 割点 (articulaion point)：删除后使得连通图不再连通的顶点
- 双连通分量：满足以下条件的边的最大子集E'
  - 由E'导出的子图是双连通的



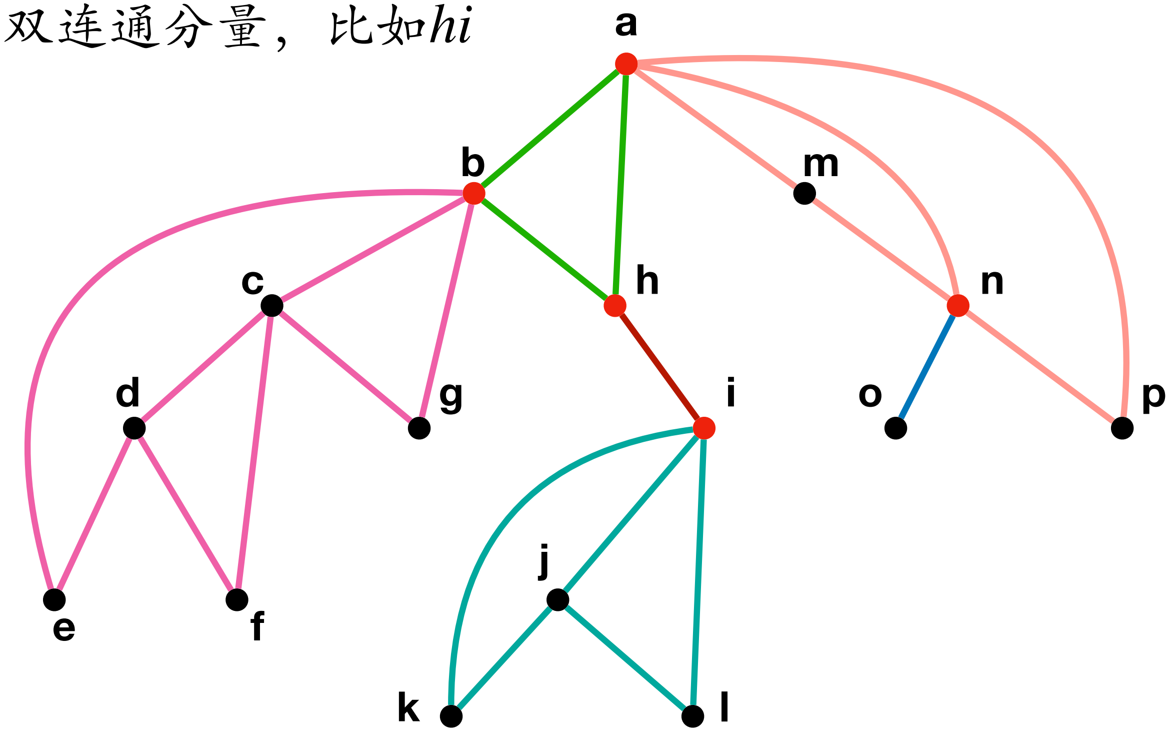
双连通图和双连通分量

- 双连通图：删除任一顶点（及相关联的边）后依然连通的无向图
  - 任意两个顶点之间存在至少两条顶点不相交的路径
- 割点 (articulaion point)：删除后使得连通图不再连通的顶点
- 双连通分量：满足以下条件的边的最大子集E'
  - 由E'导出的子图是双连通的
- 桥 (bridge)：仅包含一条边的双连通分量，比如hi



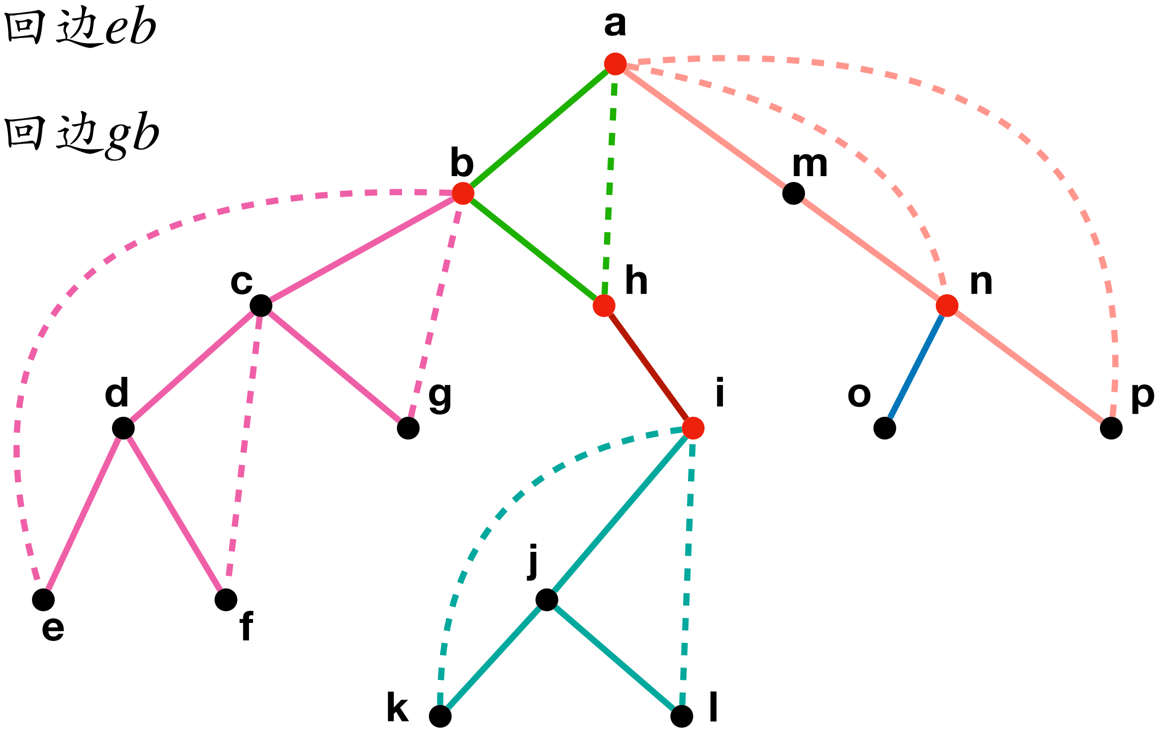
双连通图和双连通分量

- 双连通图：删除任一顶点（及相关联的边）后依然连通的无向图
  - 任意两个顶点之间存在至少两条顶点不相交的路径
- 割点 (articulaion point)：删除后使得连通图不再连通的顶点
- 双连通分量：满足以下条件的边的最大子集E'
  - 由E'导出的子图是双连通的
- 桥 (bridge)：仅包含一条边的双连通分量，比如hi
- 如何判断图G是双连通的
  - 检查所有顶点是否是割点
    - $O(n(m+n))$
    - 是否有更快的算法？
- 如何计算所有的双连通分量



割点在深搜树中的性质

- 观察：原连通分量BC中从顶点a开始深度优先搜索形成的搜索树 ← DFS树
- 顶点b是割点，因为删除b后：
  - 以c为根的子树形成一个新的连通分量
  - 虽然以h为根的子树仍然在BC中：回边ha
- 顶点c不是割点，因为删除c后：
  - 以d为根的子树仍在BC中：回边eb
  - 以g为根的子树仍在BC中：回边gb
- 顶点a是割点
  - 它有2个孩子



实线：树边 (tree edge)，虚线：回边 (back edge)

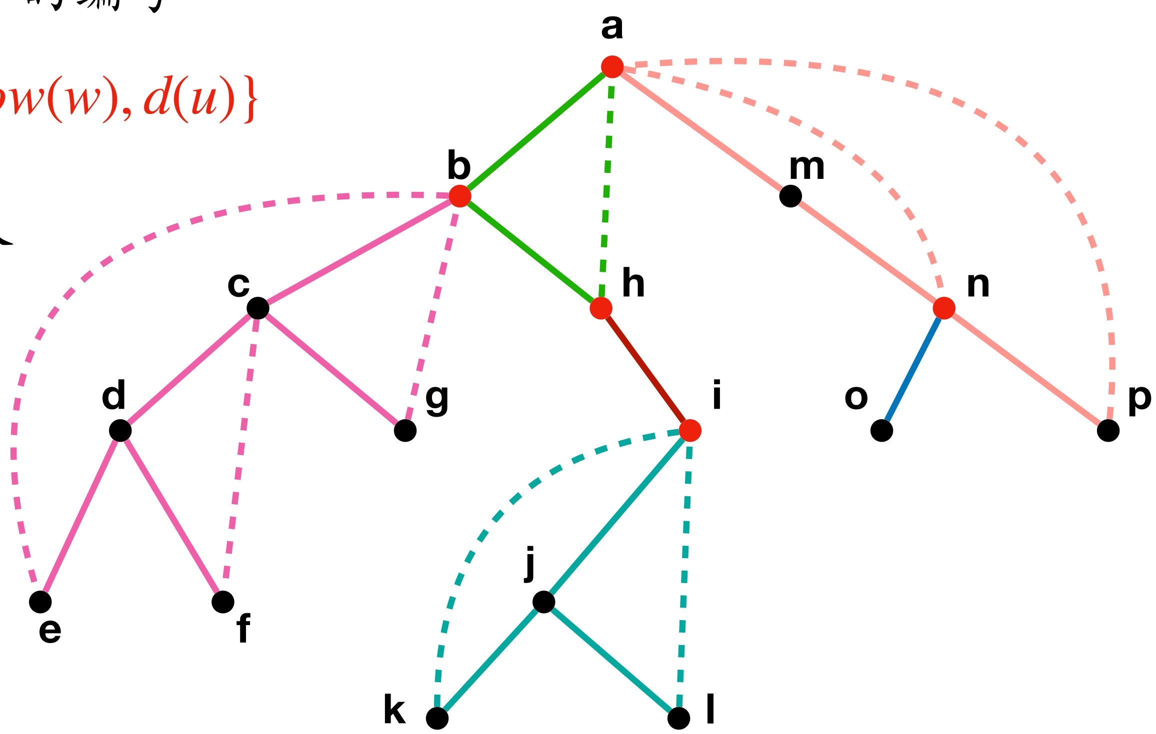
割点的判断

- DFS树的根是割点，如果它有两个或两个以上的孩子
- DFS树的一个非根顶点是割点，如果不存在一个以其孩子为根的子树，该子树有一条到“较高”顶点的回边
- 顶点在深度优先搜索序列中的编号，较高的顶点具有较小的编号
- $low(v)$：以v为根的子树通过回边所能到达的最高顶点的编号
- $d(v)$：v在深度优先搜索序列中的编号

递归关系：$low(v) = \min_{vw \in T, vu \in B} \{low(w), d(u)\}$

  - T是树边集合，B是回边集合
  - $low(v)$的初值为$d(v)$

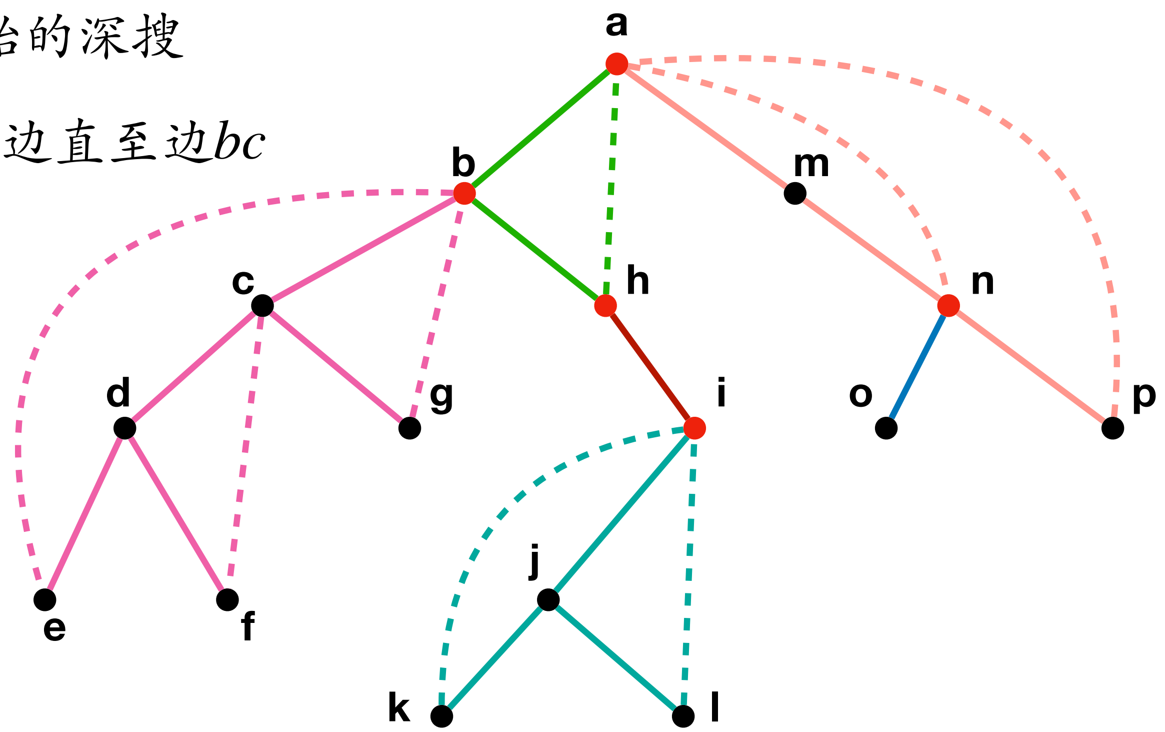
- 非根节点v是割点的条件
  - $\forall w,\ vw \in T,\ d(v) \leq low(w)$



实线：树边 (tree edge)，虚线：回边 (back edge)

双连通分量的计算

- 观察：如何发现顶点b是割点
  - 以顶点c开始的深搜结束之后，$d(b) \leq low(c)$
  - 已经访问的边：$ab, ba, bc, cb, cd, dc, de, eb, ed, df, fc, fd, cf, cg, gb, gc$
  - 删除重复的树边后：$ab, bc, cd, de, eb, df, fc, cg, gb$
  - 把所有访问的边（不包括重复的树边）放入栈中
    - 访问边bc导致以顶点c开始的深搜
    - 这次深搜结束后从栈顶的边直至边bc
    - 恰好构成一个双连通分量



实线：树边 (tree edge)，虚线：回边 (back edge)

计算双连通分量的算法

procedure dfsbcc(G, v) 时间复杂度：$O(m+n)$

Input: $G(V, E)$ is an undirected connected graph, $v \in V$

Output: all biconnected components of G

$t \leftarrow t+1$; $d(v) \leftarrow t$; $low(v) \leftarrow d(v)$; $c \leftarrow 0$

for each $vw \in E$

    if $d(w)=0$ // w has not been visited, so vw is a tree edge

        $par(w) \leftarrow v$ // v is the parent of w

        $c \leftarrow c+1$ // the number of children of v

        push vw into stack

        dfsbcc(G, w)

        $low(v)=\min\{low(v), low(w)\}$

        if $d(v)=0$ and $c>1$ or $0<d(v) \leq low(w)$ // v is an articulation point

            pop the stack down to vw // a new biconnected component

    else // w has been visited

        if $w \neq par(v)$ // and w is not v's parent, so vw is a back edge

            $low(v)=\min\{low(v), d(w)\}$

            if $d(w)<d(v)$

                push vw into stack