



武汉大学

WUHAN UNIVERSITY



算法设计与分析

图的遍历

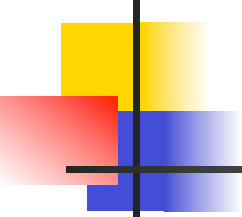
林海

Lin.hai@whu.edu.cn



- 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)



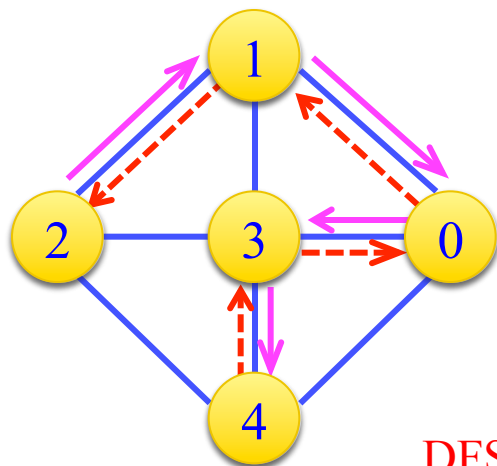
深度优先搜索 (DFS)

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 $v \in V$,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 。



深度优先遍历过程演示



$v=2$ 的DFS序列:

2 1 0 3 4

遍历过程结束



DFS思路: 距离初始顶点越远越优先访问!



深度优先搜索 (DFS)

- 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)
- **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:在调用 $\text{dfs}(v)$ 的过程中,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. $\text{predfn} \leftarrow 0; \text{postdfn} \leftarrow 0$ //计数器，在使用DFS解决某些实际问题时用到
2. for $v \in V$

3. $\text{visited}[v] \leftarrow \text{false}$

4. end for

5. for $v \in V$ //从一个顶点 v 出发，可能无法遍历全部顶点

6. if $\text{visited}[v] = \text{false}$ then $\text{dfs}(v)$

7. end for

$\text{dfs}(v)$

1. $\text{visited}[v] \leftarrow \text{true}$

2. $\text{predfn} \leftarrow \text{predfn} + 1$

3. for $(v,w) \in E$

4. if $\text{visited}[w] = \text{false}$ then $\text{dfs}(w)$

5. end for

6. $\text{postdfn} \leftarrow \text{postdfn} + 1.$

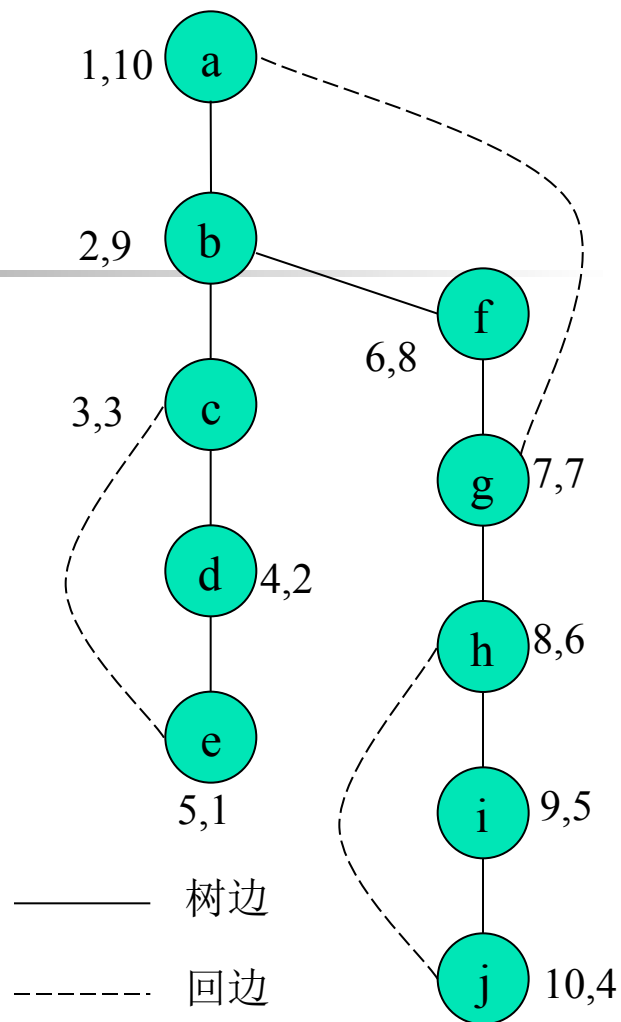
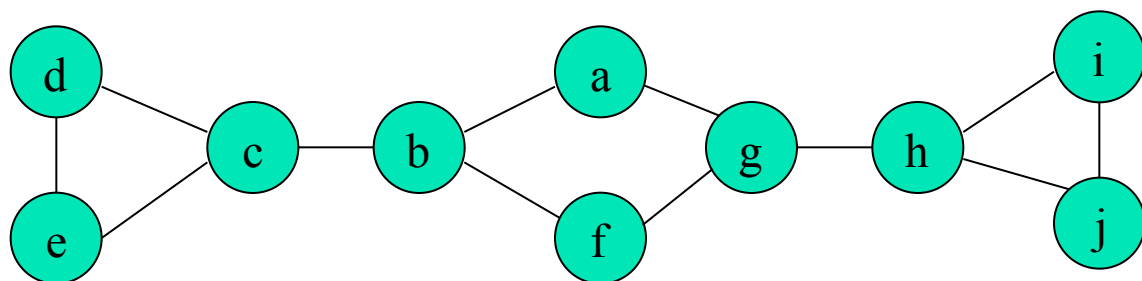
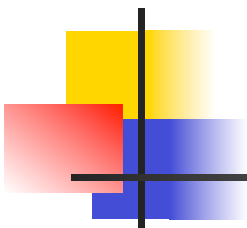
Predfn 和 postdfn
是两个计数器

- 第一个for循环是 $\Theta(n)$
- $\text{Dfs}()$ 的复杂度为 $\Theta(1)$ （不考虑for循环）， $\text{dfs}()$ 总共被调用了 n 次
- $\text{Dfs}()$ 中的for循环一共执行了 $2m$ 次（如果是有向图，则是 m 次）
- 所以总复杂度为 $\Theta(m+n)$ （操作频率）

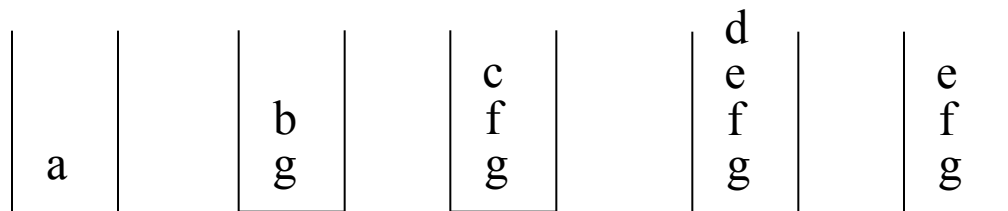
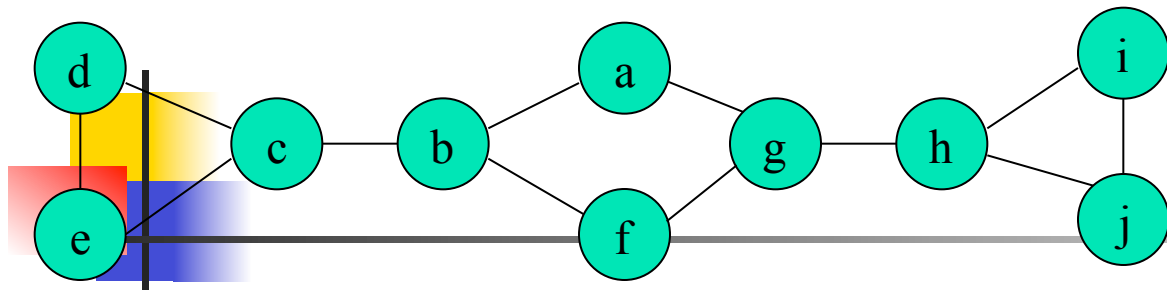


无向图情形

- 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：探测边 (v,w) 时， w 是“unvisited”状态，则边 (v,w) 是树边。
- 回边(Back edges) — G 中除却树边的所有其它边。



postdfs=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。

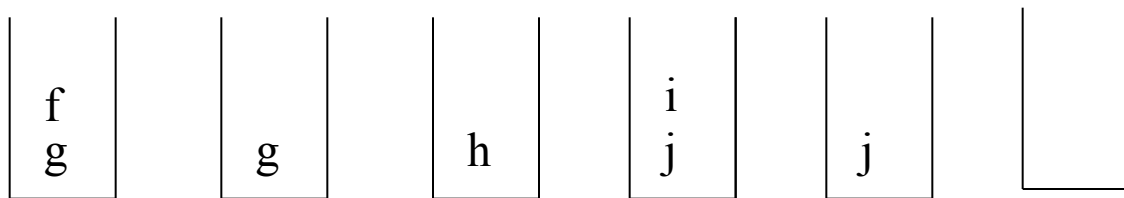


a

b

c

d



e

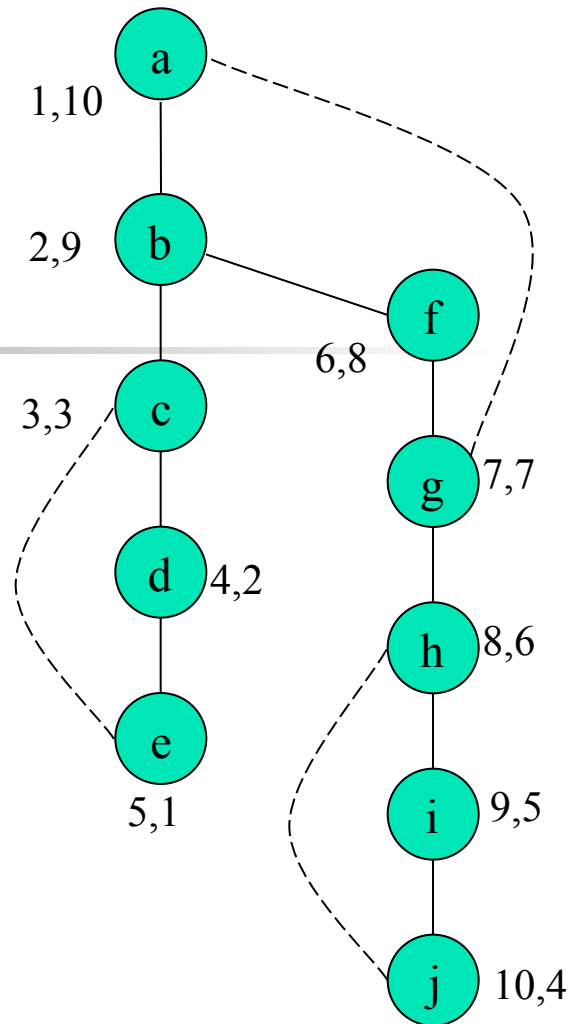
f

g

h

i

j



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

递归形式

dfs(v)

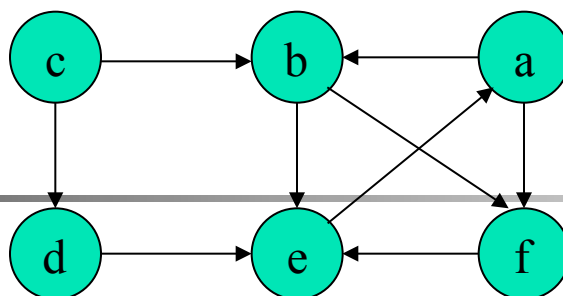
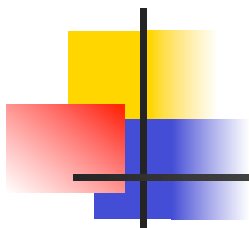
1. Push(v,S)
2. visited[v] \leftarrow true
3. While S \neq {}
4. v \leftarrow Pop(S) // visit
5. for (v,w) \in E
6. if visited[w] = false then
7. Push(w,S)
8. visited[w] = true
9. end if
10. end for
11. end while

迭代形式



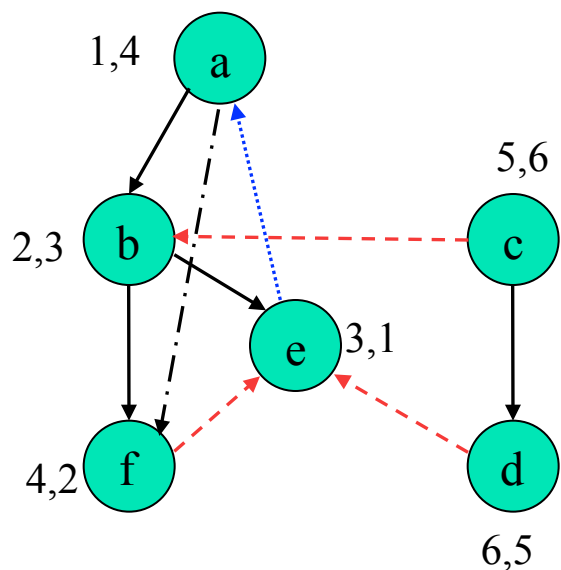
有向图情形

- 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（由不互为祖先的节点组成）。

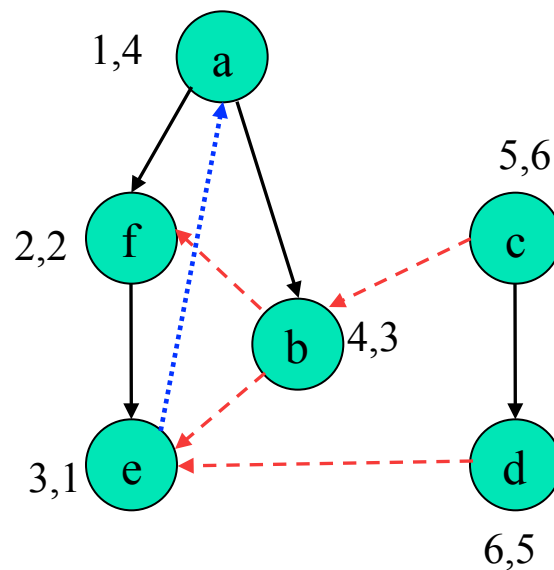


——> 树边 -.-.-> 前向边 > 回边 - - - -> 横跨边

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





为何DFS用于无向图时，不存在前向边及横跨边？

(1) 前向边 (v, w)

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。

(2) 横跨边 (w, v)

假设存在横跨边，则存在 v, w 都不互为祖先，且 w 在 v 之前被访问，否则 (v, w) 将成树边；由于访问 w 时，未访问到 v ，故 (w, v) 不存在，矛盾。故无横跨边。



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
- 强连通分支
- 网络页面检索



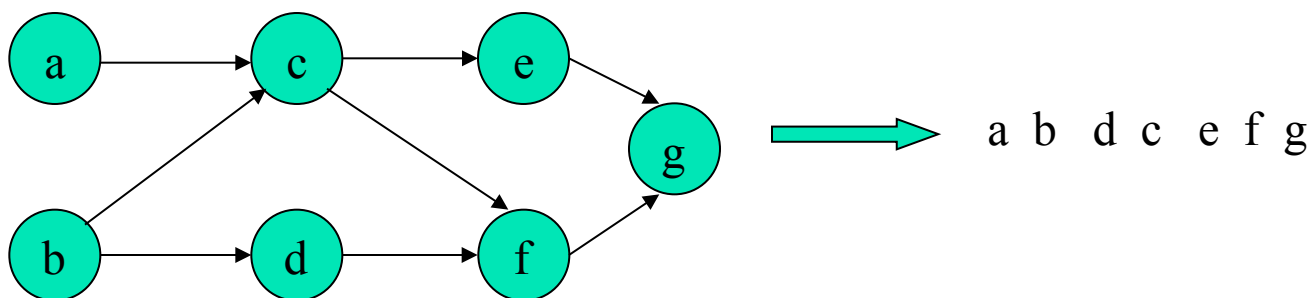
图回路判定

- 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 $(v,w) \in E$, 那么, 在线性序中, v 在 w 之前出现。

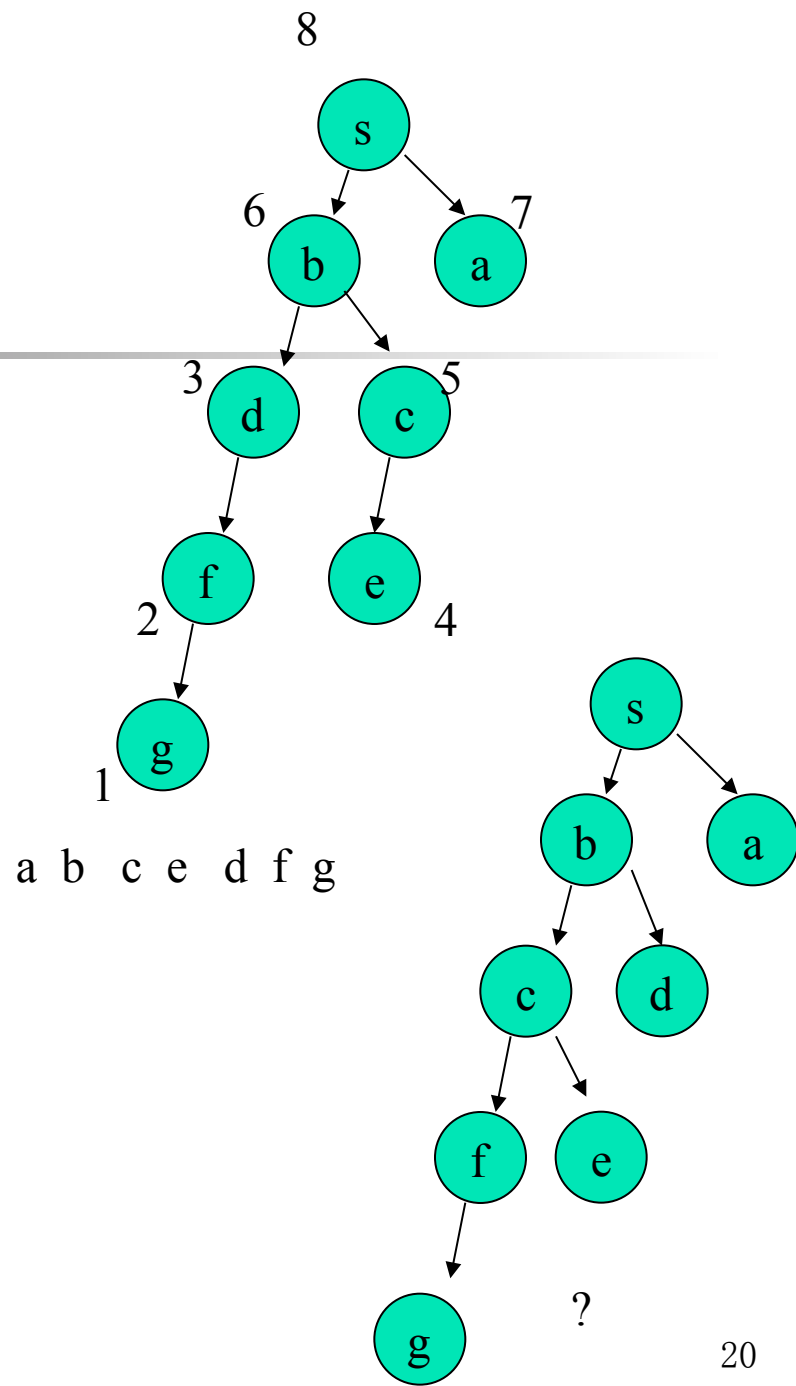
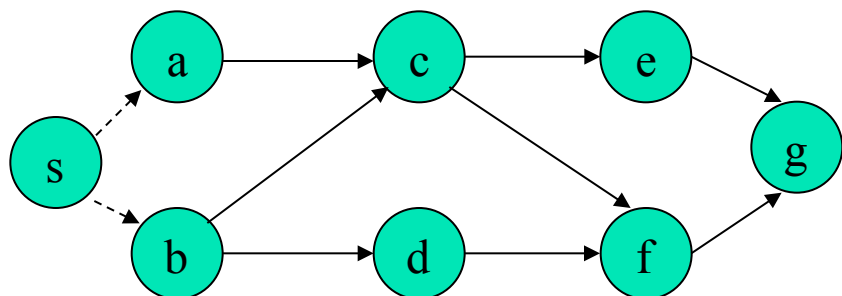
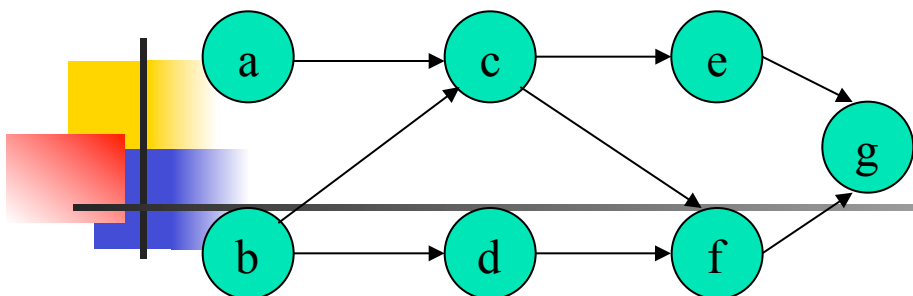


- 我们假设在DAG中只有唯一一个入度为0的顶点; 如果有一个以上的顶点入度为0, 可以通过添加一个新的顶点 s , 然后将 s 指向所有入度为0的顶点, 这样 s 就成为唯一一个入度为0的顶点。



拓扑排序(Topological sorting)

- 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。





寻找关节点(Finding articulation points in a graph*)

- 关节点: 给定无向图 $G=(V,E)$, $|V|>2$ 。
称 $v \in G$ 为一关节点, 当且仅当存在另外两个不同的顶点 $u \in G$ 和 $w \in G$, 并且 u 与 v 之间的任意路径均必须经由 v 通过。
 - 显然, 如果 G 是连通的, 那么在移除关节点和与其关联的边后, 图变为不连通的。



寻找关节点

- 深度优先遍历G
- 标记两个标号： $\alpha[v]$ 和 $\beta[v]$
 - $\alpha[v] := \text{predfn}$, 每次dfs中加1
 - $\beta[v]$: 初始化= $\alpha[v]$, dfs中为下面几个中的最小值
 - $\alpha[v]$
 - $\alpha[u]$, 对于每个顶点u, (v,u) 是回边
 - $\beta[w]$, 在深度优先搜索树中的每条边 (v,w)



寻找关节点

■ 确定关节点

- 根是一个关节点，当且仅当在深度优先搜索树中，它有两个或更多的儿子。
- 根以外顶点 v 是关节点，当且仅当 v 有一个儿子 w ，使得 $\beta[w] \geq \alpha[v]$



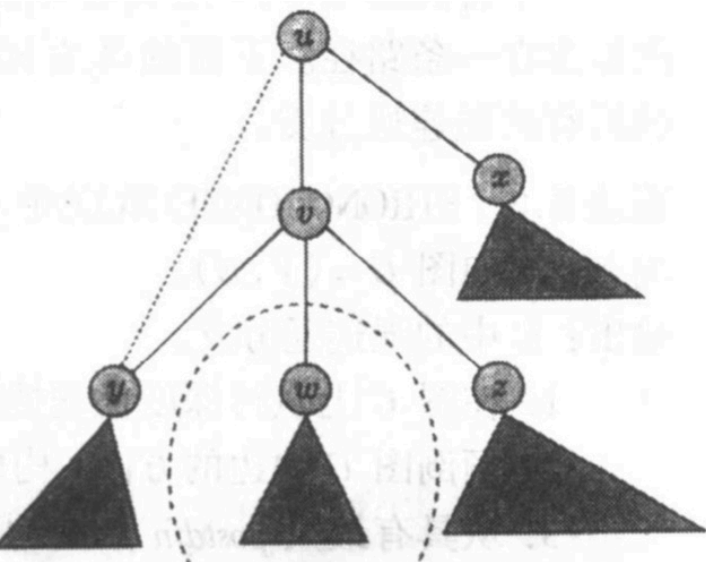
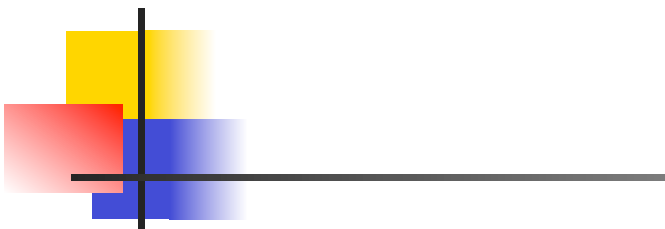
算法

算法 9.2 ARTICPOINTS

输入：连通无向图 $G = (V, E)$ 。

输出：包含 G 的所有可能关节节点的数组 $A[1 \cdots count]$ 。

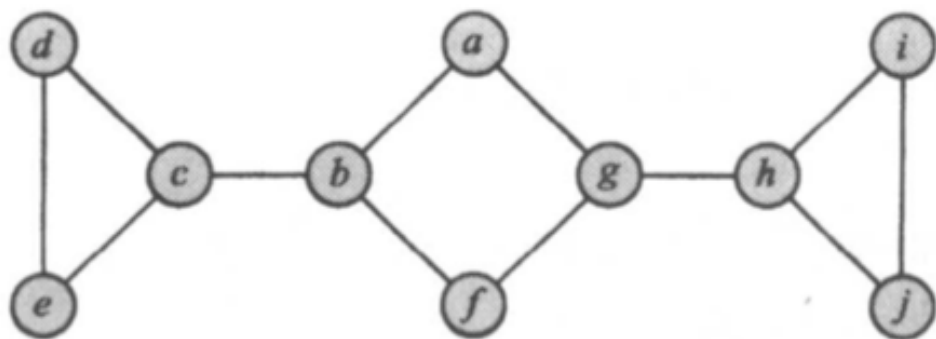
1. 设 s 为起始顶点
2. **for** 每个顶点 $v \in V$
3. 标记 v 未访问
4. **end for**
5. $predfn \leftarrow 0$; $count \leftarrow 0$; $rootdegree \leftarrow 0$
6. $dfs(s)$



1. 标记 v 已访问; $artpoint \leftarrow \text{false}$; $predfn \leftarrow predfn + 1$
2. $\alpha[v] \leftarrow predfn$; $\beta[v] \leftarrow predfn$ {初始化 $\alpha[v]$ 和 $\beta[v]$ }
3. **for** 每条边 $(v, w) \in E$
4. **if** (v, w) 为树边 **then**
5. $dfs(w)$
6. **if** $v = s$ **then**
7. $rootdegree \leftarrow rootdegree + 1$
8. **if** $rootdegree = 2$ **then** $artpoint \leftarrow \text{true}$
9. **else**
10. $\beta[v] \leftarrow \min\{\beta[v], \beta[w]\}$
11. **if** $\beta[w] \geq \alpha[v]$ **then** $artpoint \leftarrow \text{true}$
12. **end if**
13. **else if** (v, w) 是回边 **then** $\beta[v] \leftarrow \min\{\beta[v], \alpha[w]\}$
14. **else do nothing** { w 是 v 的父亲 }
15. **end if**
16. **end for**
17. **if** $artpoint$ **then**
18. $count \leftarrow count + 1$
19. $A[count] \leftarrow v$
20. **end if**



寻找关节点



—— 树边 回边

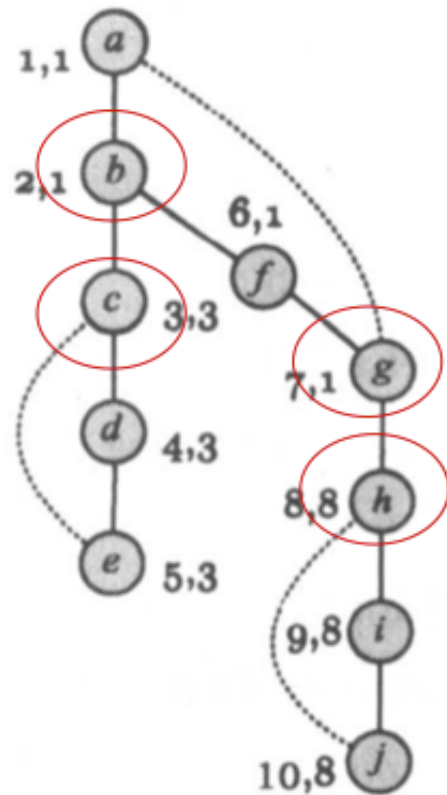


图 9.5 在图中查找关节点的例子



强连通分支

- 有向图 $G=(V,E)$ ，强连通集为顶点的极大集
- 在该集合中，每一对顶点都存在一条路径



算法

算法 9.3 STRONGCONNECTCOMP

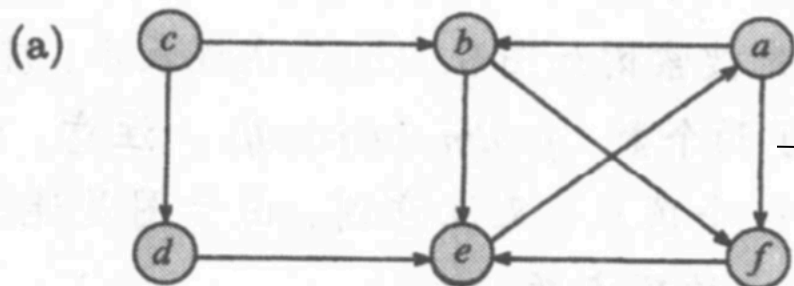
输入：有向图 $G = (V, E)$ 。

输出： G 中的强连通分支。

1. 在图 G 上执行深度优先搜索，对每一个顶点赋给相应的 *postdfn* 值。
2. 颠倒图 G 中边的方向，构成一个新的图 G' 。
3. 从具有最大 *postdfn* 数值的顶点开始，在 G' 上执行深度优先搜索，如果深度优先搜索不能到达所有的顶点，则在余下的顶点中找一个 *postdfn* 数值最大的顶点，开始下一次深度优先搜索。
4. 在最终得到的森林中的每一棵树对应一个强连通分支。

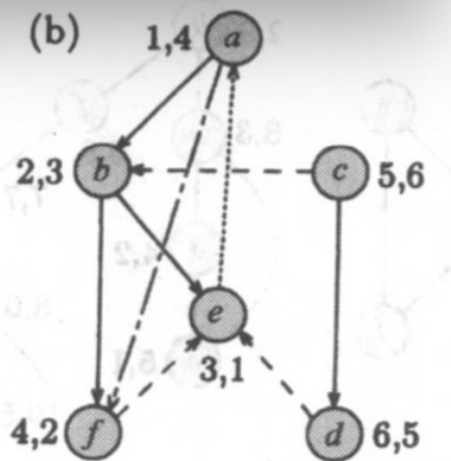


举例

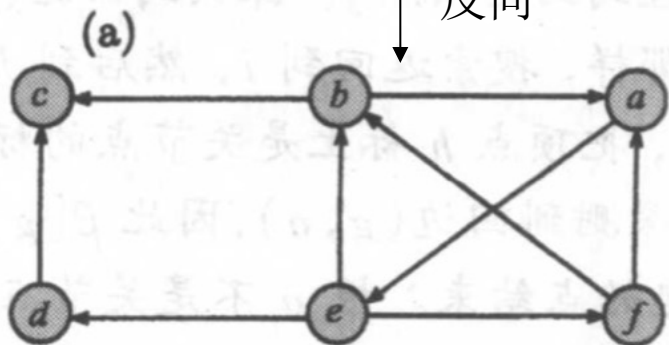


—— 树边 - - - 前向边
- · - · 回边 - - - 横跨边

DFS

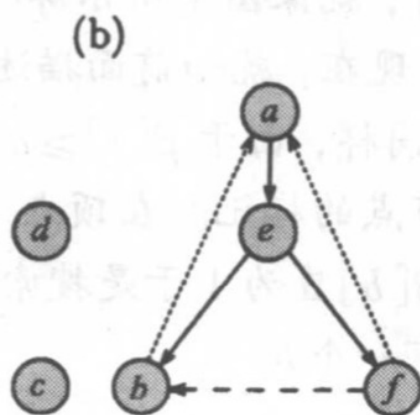


反向



—— 树边 - · - · 回边
- - - 横跨边

DFS





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。

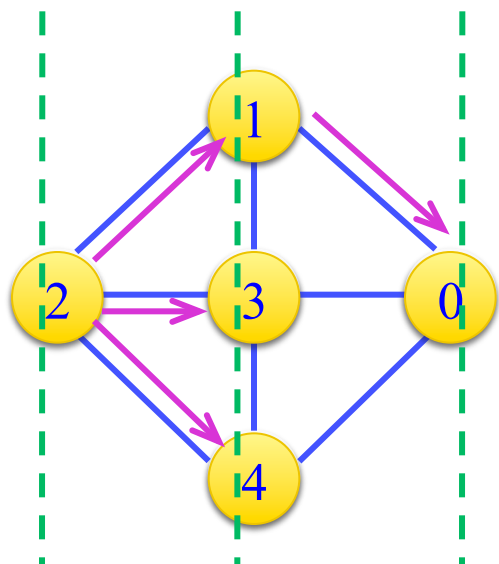


广度优先搜索(Breadth-First Search, BFS)

- 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。



广度优先遍历过程演示



$v=2$ 的BFS序列:

2 1 3 4 0

遍历过程结束



BFS思路: 距离初始顶点越近越优先访问!



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)

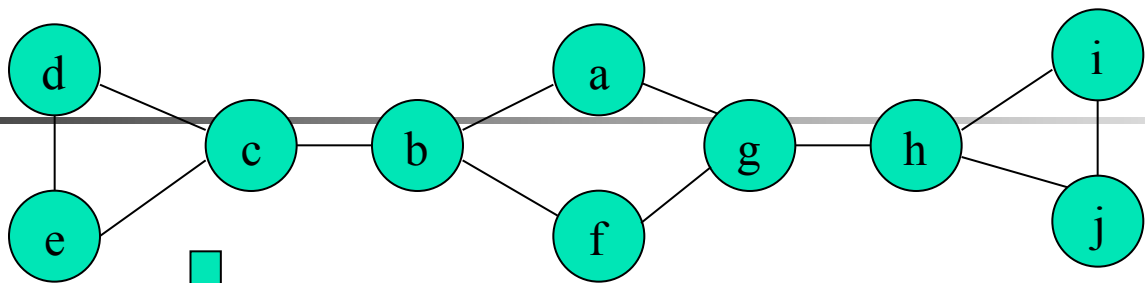
1. $bf_n \leftarrow 0$
2. for $v \in V$
3. $visited[v] \leftarrow false$
4. end for
5. for $v \in V$
6. if $visited[v] = false$ then $bfs(v)$
7. end for

$bfs(v)$

1. $Q \leftarrow \{v\}$
2. $visited[v] \leftarrow true$
3. while $Q \neq \{\}$ //to be visited
4. $v \leftarrow pop(Q)$
5. $bf_n \leftarrow bf_n + 1$
6. for $(v,w) \in E$
7. if $visited[w] = false$ then
8. $Push(w,Q)$
9. $visited[w] \leftarrow true$
10. end if
11. end for
12. end while

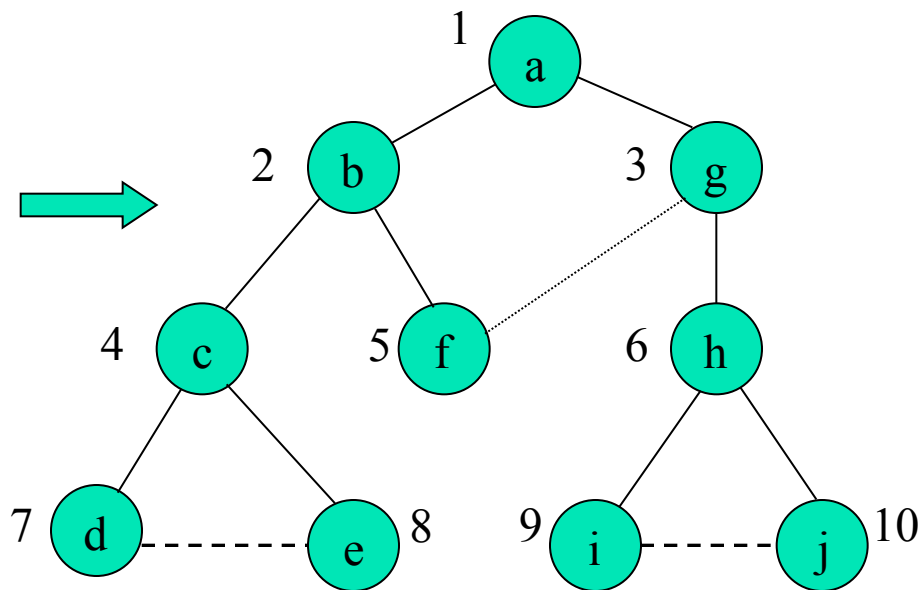
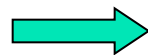


无向图BFS实例



Queue

a	b	g	c	f	h	d	e	i	j
---	---	---	---	---	---	---	---	---	---

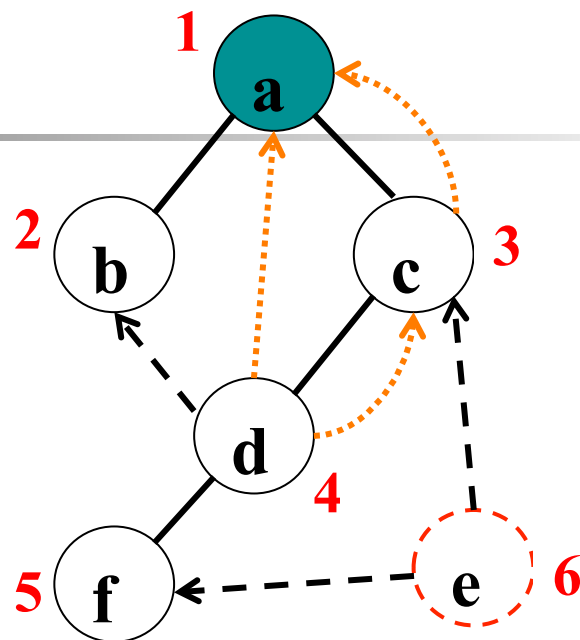
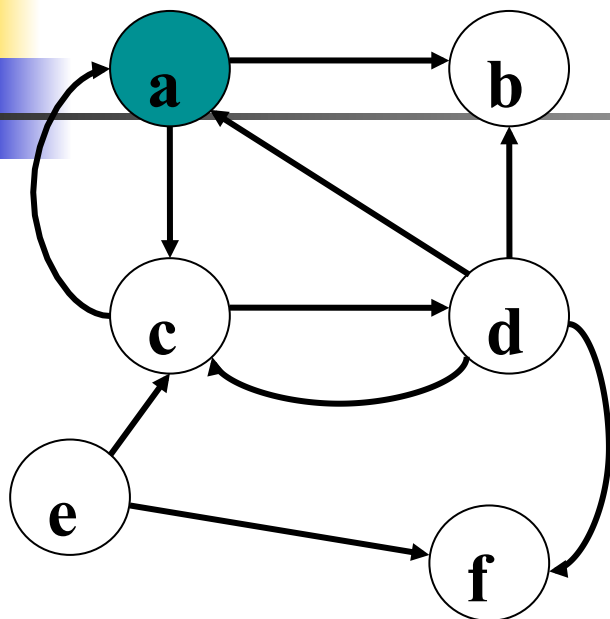


—— 树边
----- 横跨边

时间复杂度: $\Theta(m+n)$



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。



广度优先搜索应用

- $G=(V,E)$, s 是 V 中一个顶点
- 以 S 作为起始点时，产生的广度优先搜索树中从 s 点到其他任意顶点的路径有最少的边数。
- 问题：找出从 s 到其他每一顶点的距离，这里从 s 到一个顶点 v 的距离定义为：从 s 到任意路径的最少边数



思路

- 在每个顶点压入队列前，标上它的距离就可以了。
- 这样，起始点将标上0,它的邻接点是1,依次类推。很明显，
- 每个顶点的标号是它到起始点的最短距离。



练习

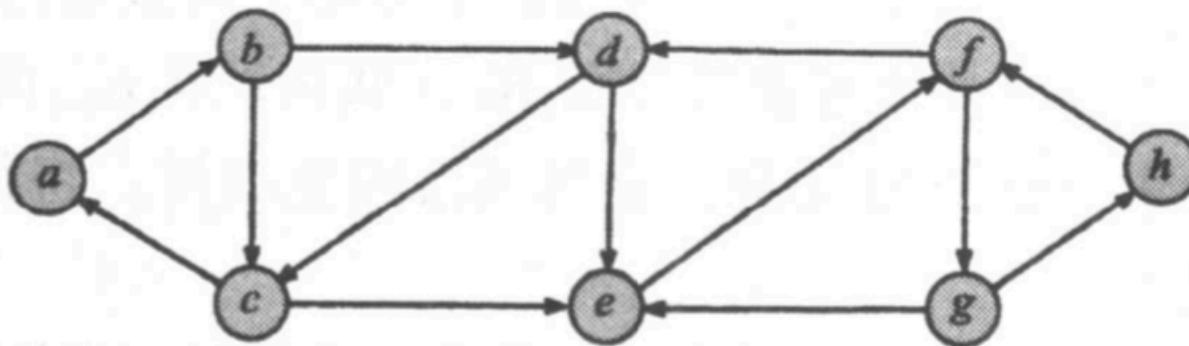


图 9.10 一个有向图

1. 从顶点e开始运行深度优先搜索的结果，并给出边的分类
2. 从顶点e开始运行广度优先搜索的结果，并给出边的分类
3. 哪个是关节点（图为无向图）？
4. 应用强连通分支的算法



作业

p170: 9.2, 9.9, 9.13, 9.16, 9.20