



# 基本的图算法

---

湖南大学信息科学与工程学院

# 提纲



1 图的基本概念

2 广度优先搜索

3 深度优先搜索

4 深度优先搜索应用

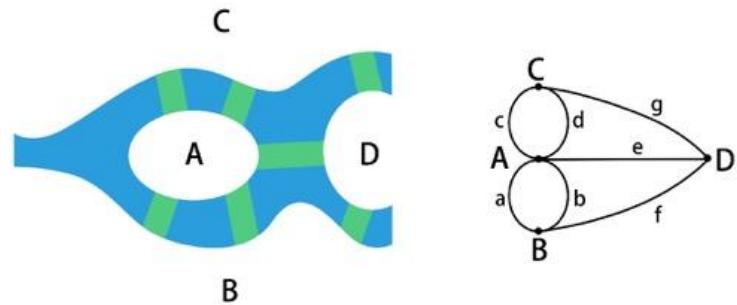


# 图论的起源

18世纪哥尼斯堡城的普莱格尔河上有七座桥，能否一次走遍七座桥，并且每座桥只允许通过一次，最后仍可以回到起始地点？

1736年，欧拉运用数学抽象法将其转换为一笔画问题，并论述和证明了该过程是绝对不可能实现的。由此图论诞生。欧拉也成为图论的创始人。

哥尼斯堡七桥图



欧拉





# 图的定义

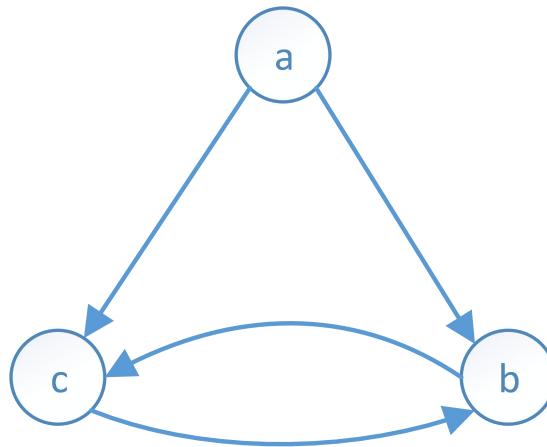
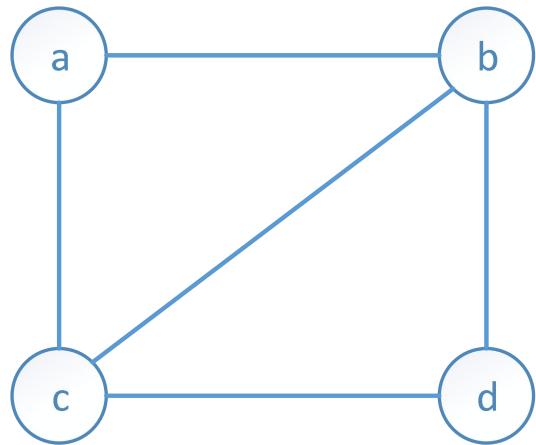
这里所说的图，是以一种抽象的形式来表示若干对象结合以及这些对象之间的关系

图可以表示成一个二元组  $G = \langle V, E \rangle$ ，其中  $V$  是点集合， $E \subseteq V \times V$  表示边集合，其中每条边由  $V$  中两个点相连接所构成



# 无向图和有向图

- 如果图中的边都没有方向，那么  $G$  被称为无向图
- 如果图中的边有方向，那么  $G$  被称为有向图





# 图上的路径

给定一个无向图  $G = (V, E)$ ,  $G$  中顶点与边的交替序列  $\Gamma = v_0 e_1 v_1 e_2 \dots e_l v_l$  称为点  $v_0$  到点  $v_l$  的路径, 其中  $v_{r-1}$  和  $v_r$  是  $e_r$  的端点 ( $1 \leq r \leq l$ )

有向图中路径的定义与无向图中定义类似, 只是要注意在有向图中通路和回路中边的方向的一致性, 即  $v_{r-1}$  必须是  $e_r$  的起点且  $v_r$  是  $e_r$  的终点 ( $1 \leq r \leq l$ )

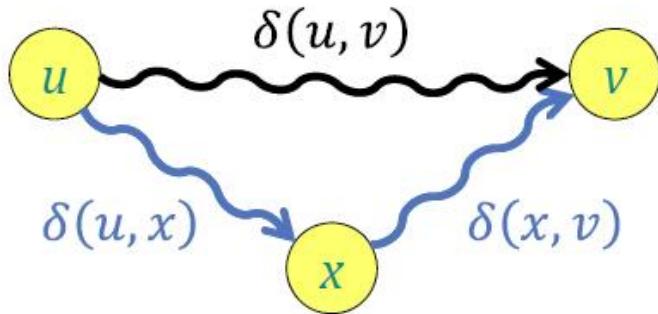
所有  $v_0$  和点  $v_l$  之间的路径中最短的那条路径称为最短路径, 最短路径的长度被称为  $v_0$  到点  $v_l$  的距离, 记为  $\delta(v_0, v_l)$

如果不存在的  $v_0$  到  $v_l$  路径, 那么  $\delta(v_0, v_l)$  记为  $\infty$



# 三角不等式

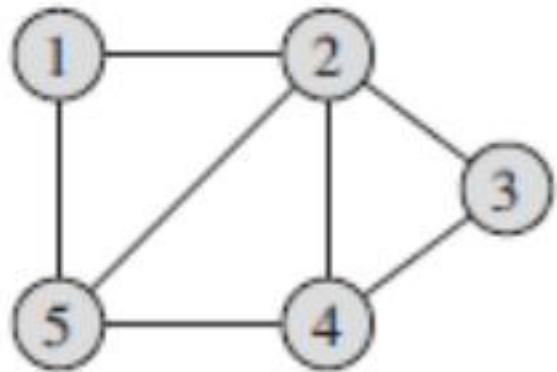
对于图上任意三个点 $u$ 、 $v$ 和 $x$ 而言， $\delta(u, v) \leq \delta(u, x) + \delta(x, v)$



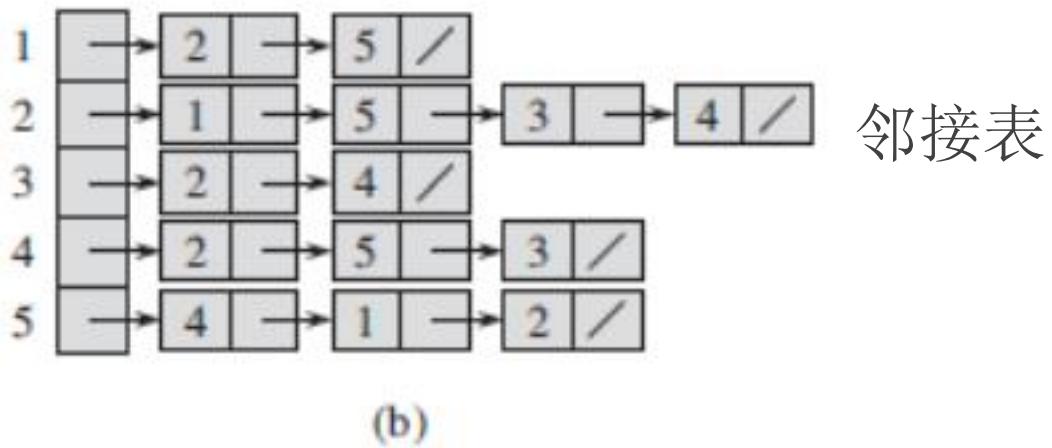
推论：对于图上任意三个点 $u$ 、 $v$ 和 $x$ 且存在边 $(x,v)$ 而言， $\delta(u, v) \leq \delta(u, x) + 1$



# 无向图的表示



(a)

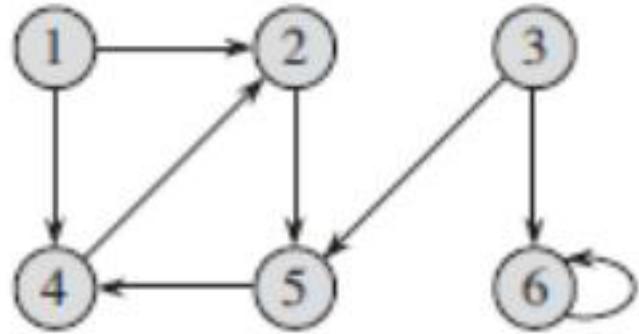


	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

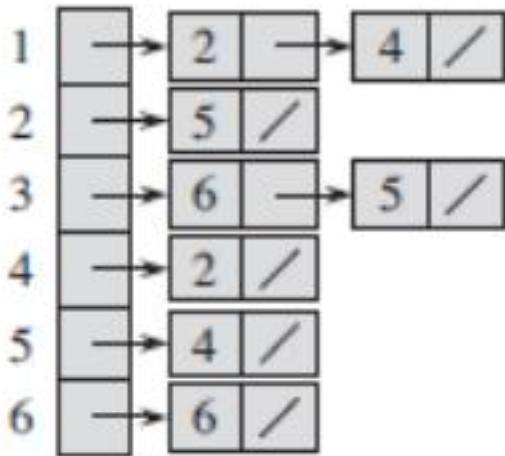
(c)

邻接矩阵

# 有向图的表示



(a)



(b)

	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

(c)

邻接表

邻接矩阵

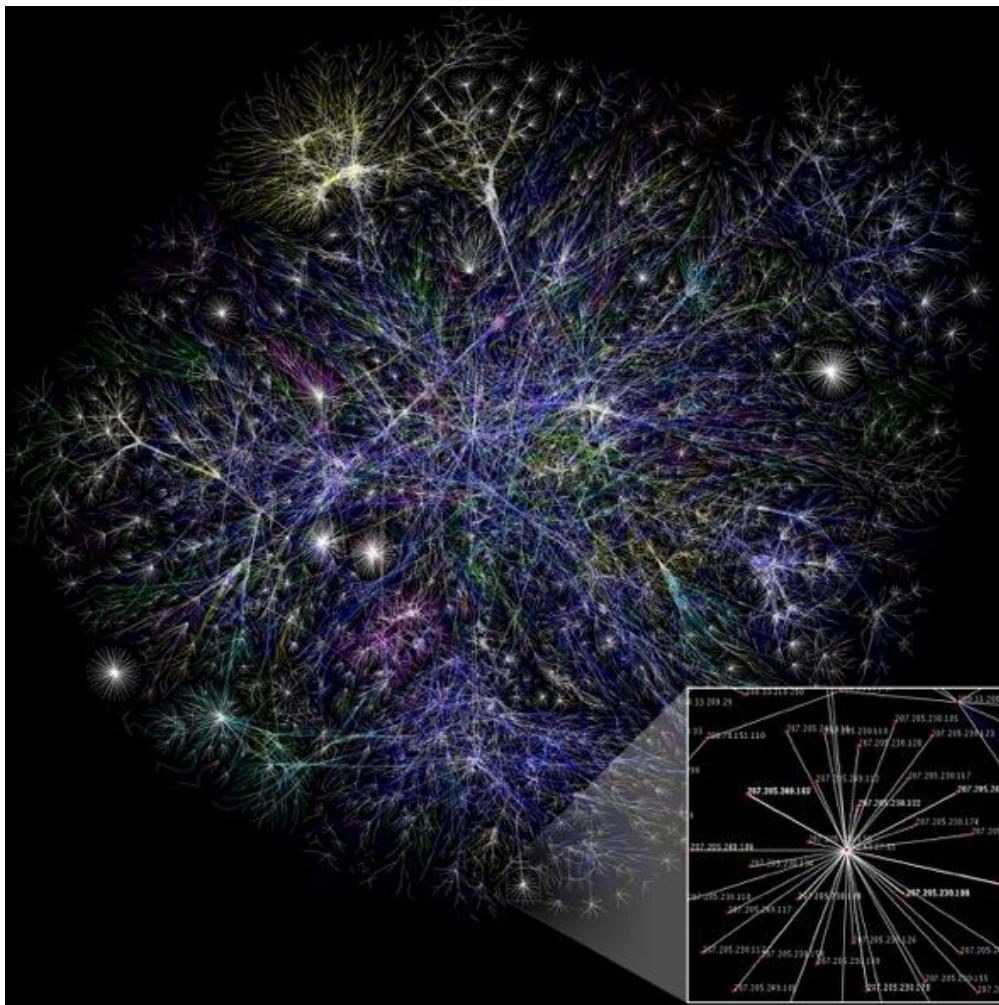
# 图的应用：网页分析



## 网页分析

- 有向网络
  - 节点：Web页面
  - 边：超链接

2003年，Barrett Lyon将400多万个网页绘制成互联网图





# 图的应用：交通网络

交通网络（美国航线）

如何从一个城市到另一个城市？

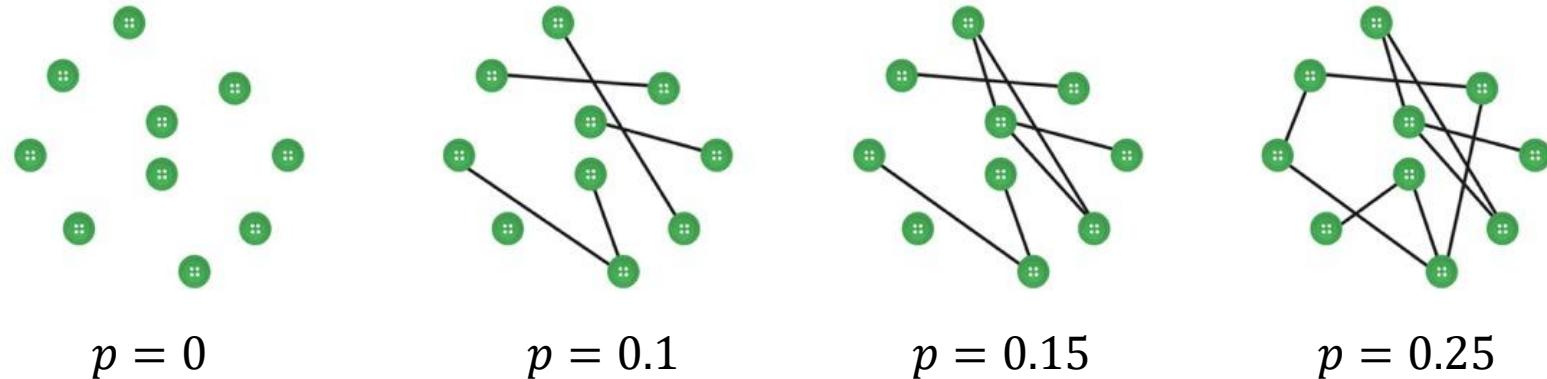




# 图的应用：随机图

著名的匈牙利数学家Renyi和Erdos在20世纪60年代提出了随机图理论（random graph theory）。

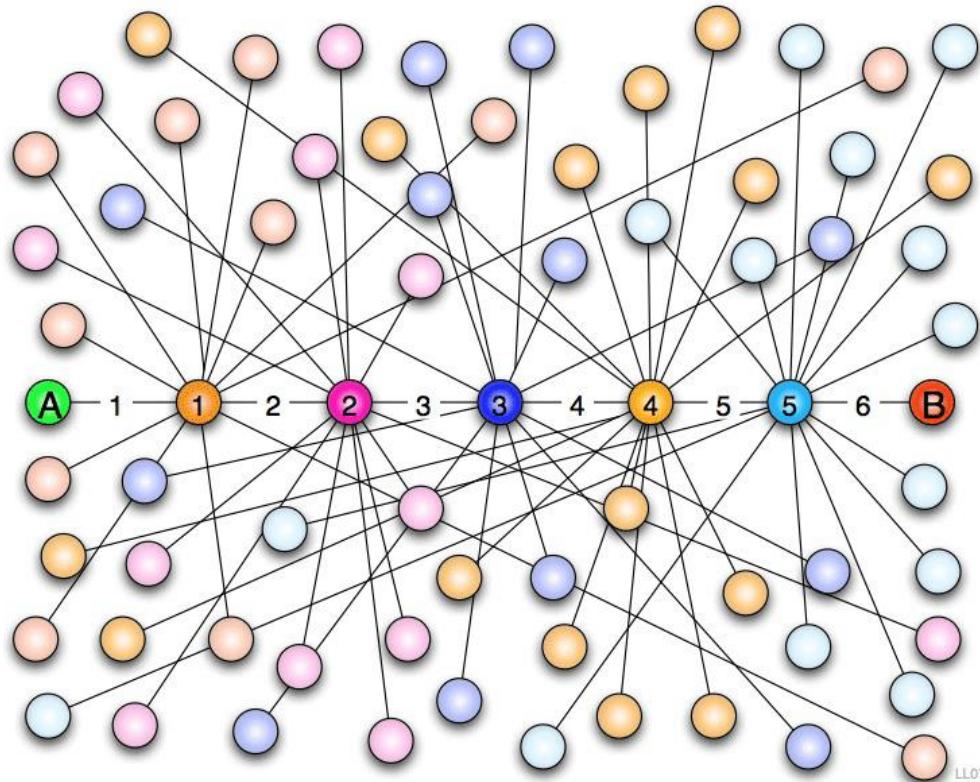
他们提出了ER随机图模型：在给定 $\diamond$ 个顶点后，规定每两个顶点之间都有 $\diamond$ 的概率连起来（ $0 \leq \diamond \leq 1$ ），而且这些判定之间两两无关。





# 图的应用：小世界理论

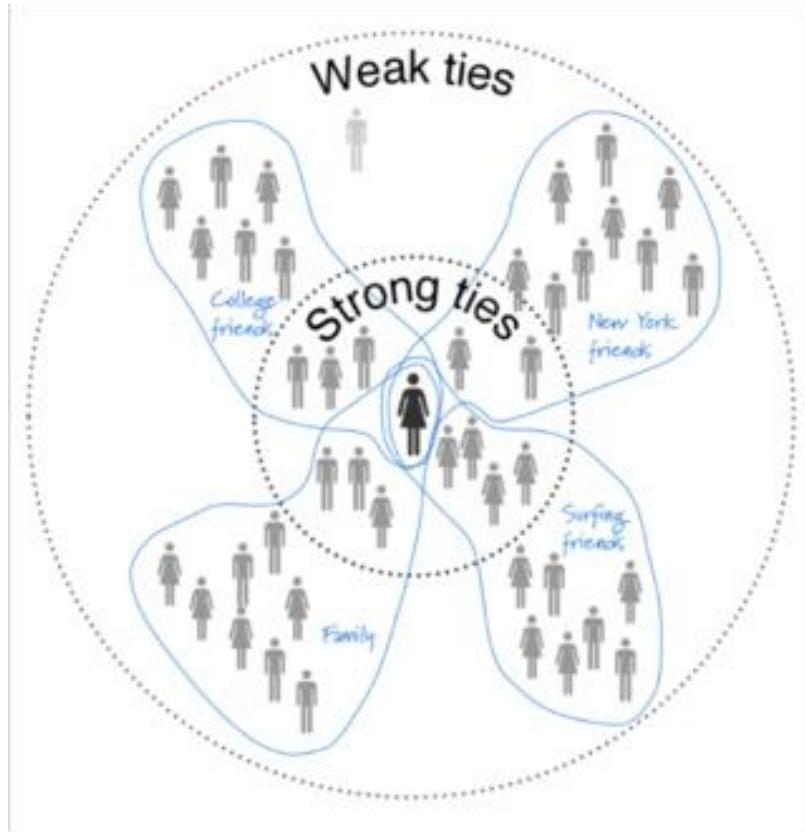
六度分离哈佛大学著名的社会心理学家Stanley Milgram在20世纪60年代给出推断：在地球上的任意两个人的平均距离是6。即平均只要通过中间的5个人，你就可以与地球上任意一个人发生联系。





# 弱连接的强度

斯坦福大学社会学教授 Mark Granovetter 在1973年发现：在找工作、找对象等等机会的面前，与我们交情比较浅的人（具有弱连接性），往往能给我们带来最大的转变。



# 提纲



1 图的基本概念

2 广度优先搜索

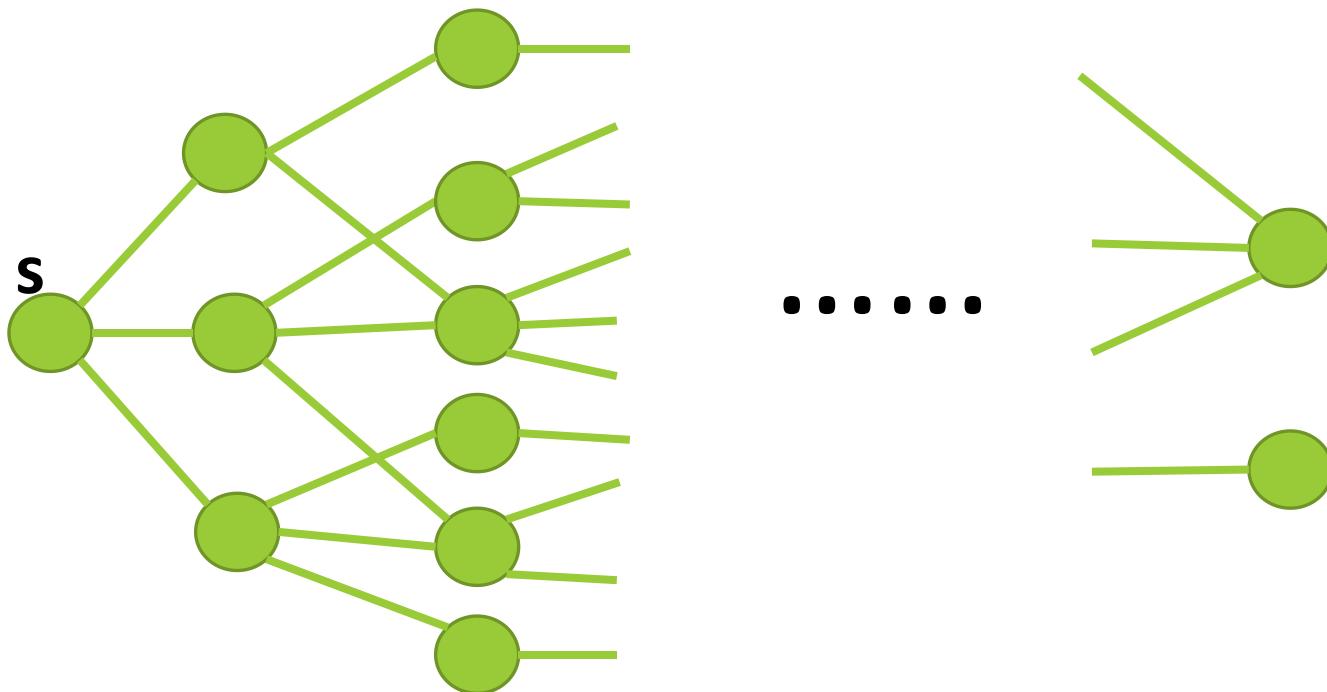
3 深度优先搜索

4 深度优先搜索应用



# 2 广（宽）度优先搜索

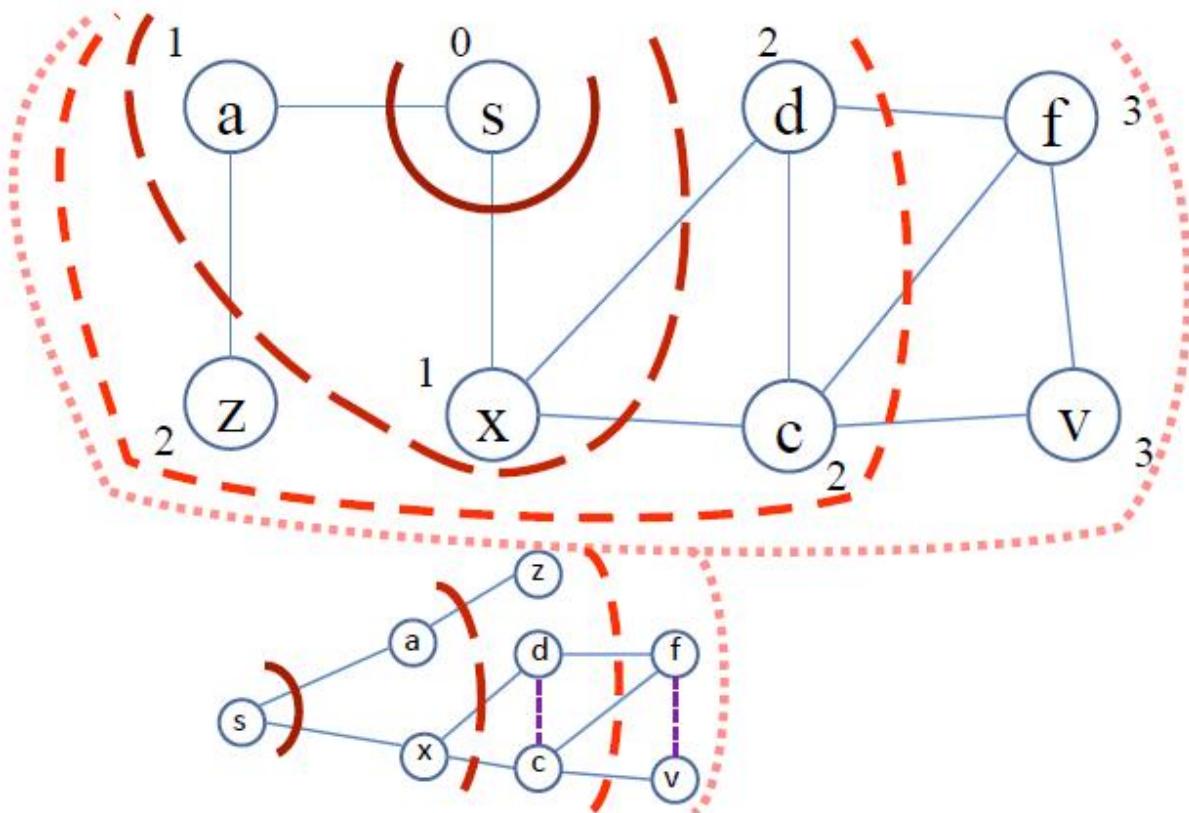
图搜索（Graph Search），求图上从源点出发到其他点的路径。





## 2 广（宽）度优先搜索

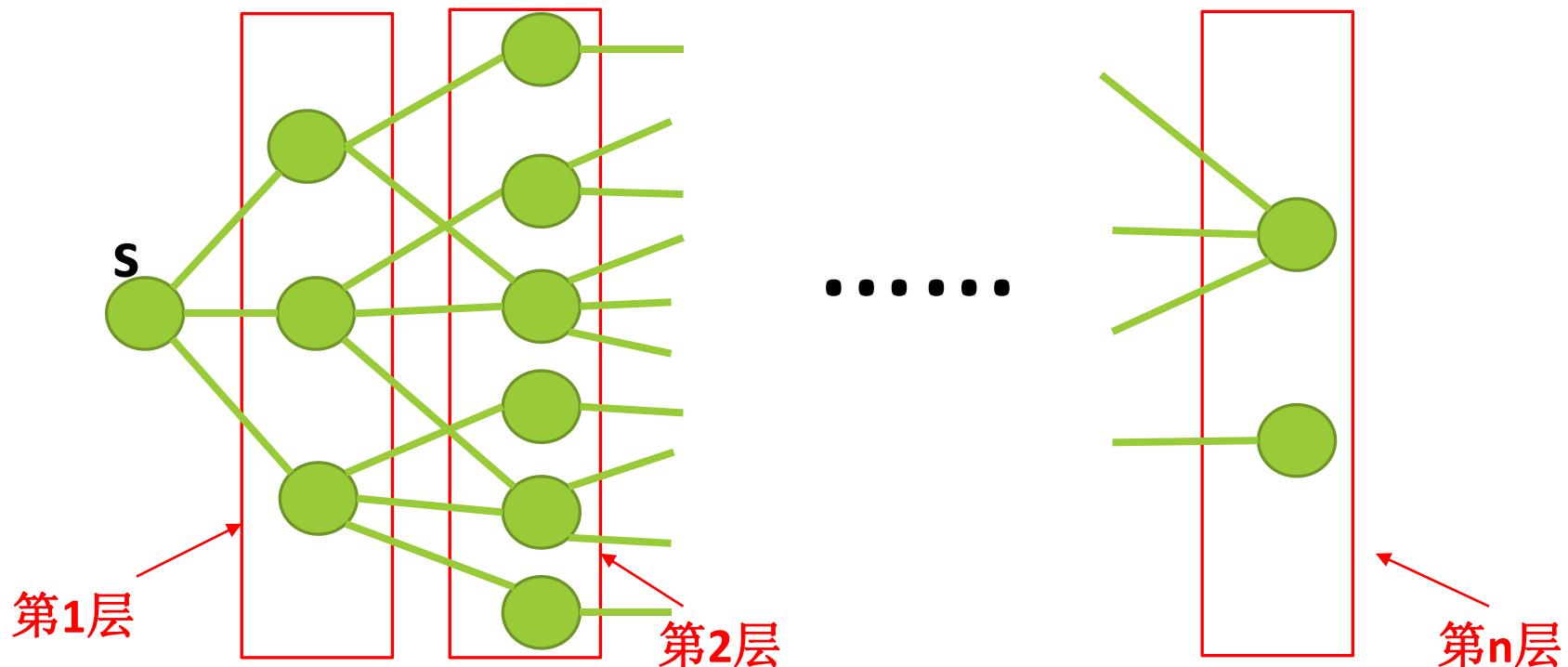
广度优先搜索（BFS,Breadth First Search），从初始点开始，逐层向前搜索，即第n层搜索未完成，不得进入下一层搜索。





# 2 广（宽）度优先搜索

广度优先搜索（BFS,Breadth First Search），从初始点开始，逐层向前搜索，即第n层搜索未完成，不得进入下一层搜索。



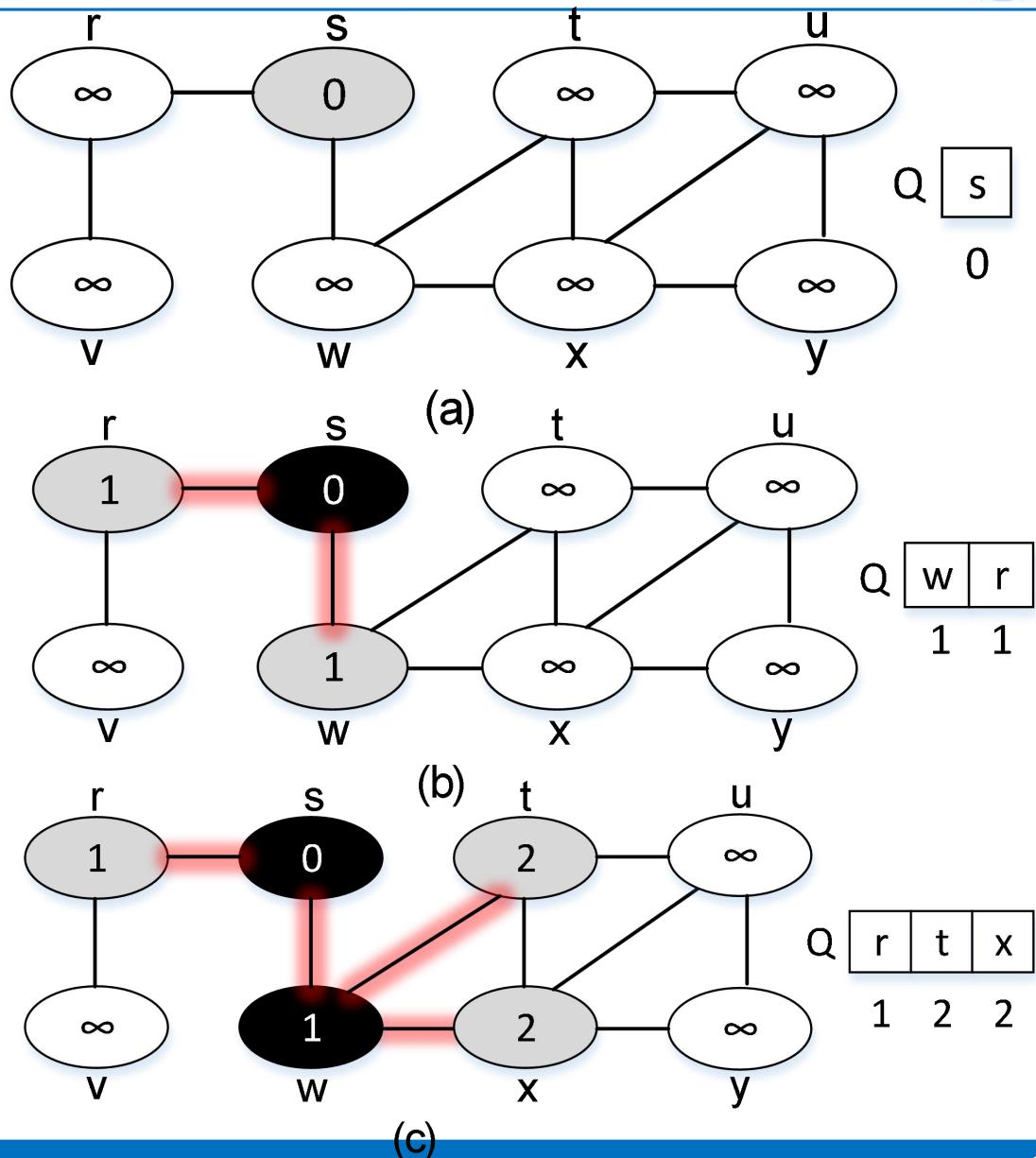
# 2 广度优先搜索-BFS实现过程



BFS( $G, s$ )

```

1 for each vertex  $u \in G.V - \{s\}$ 
2    $u.color = \text{WHITE}$ 
3    $u.d = \infty$ 
4    $u.\pi = \text{NIL}$ 
5    $s.color = \text{GRAY}$ 
6    $s.d = 0$ 
7    $s.\pi = \text{NIL}$ 
8    $Q = \emptyset$ 
9   ENQUEUE( $Q, s$ )
10  while  $Q \neq \emptyset$ 
11     $u = \text{DEQUEUE}(Q)$ 
12    for each  $v \in G.Adj[u]$ 
13      if  $v.color == \text{WHITE}$ 
14         $v.color = \text{GRAY}$ 
15         $v.d = u.d + 1$ 
16         $v.\pi = u$ 
17        ENQUEUE( $Q, v$ )
18     $u.color = \text{BLACK}$ 
```





## 2 广度优先搜索-BFS实现过程

BFS( $G, s$ )

```
1  for each vertex  $u \in G.V - \{s\}$ 
2       $u.color = \text{WHITE}$ 
3       $u.d = \infty$ 
4       $u.\pi = \text{NIL}$ 
5   $s.color = \text{GRAY}$ 
6   $s.d = 0$ 
7   $s.\pi = \text{NIL}$ 
8   $Q = \emptyset$ 
9  ENQUEUE( $Q, s$ )
10 while  $Q \neq \emptyset$ 
11      $u = \text{DEQUEUE}(Q)$ 
12     for each  $v \in G.Adj[u]$ 
13         if  $v.color == \text{WHITE}$ 
14              $v.color = \text{GRAY}$ 
15              $v.d = u.d + 1$ 
16              $v.\pi = u$ 
17             ENQUEUE( $Q, v$ )
18      $u.color = \text{BLACK}$ 
```

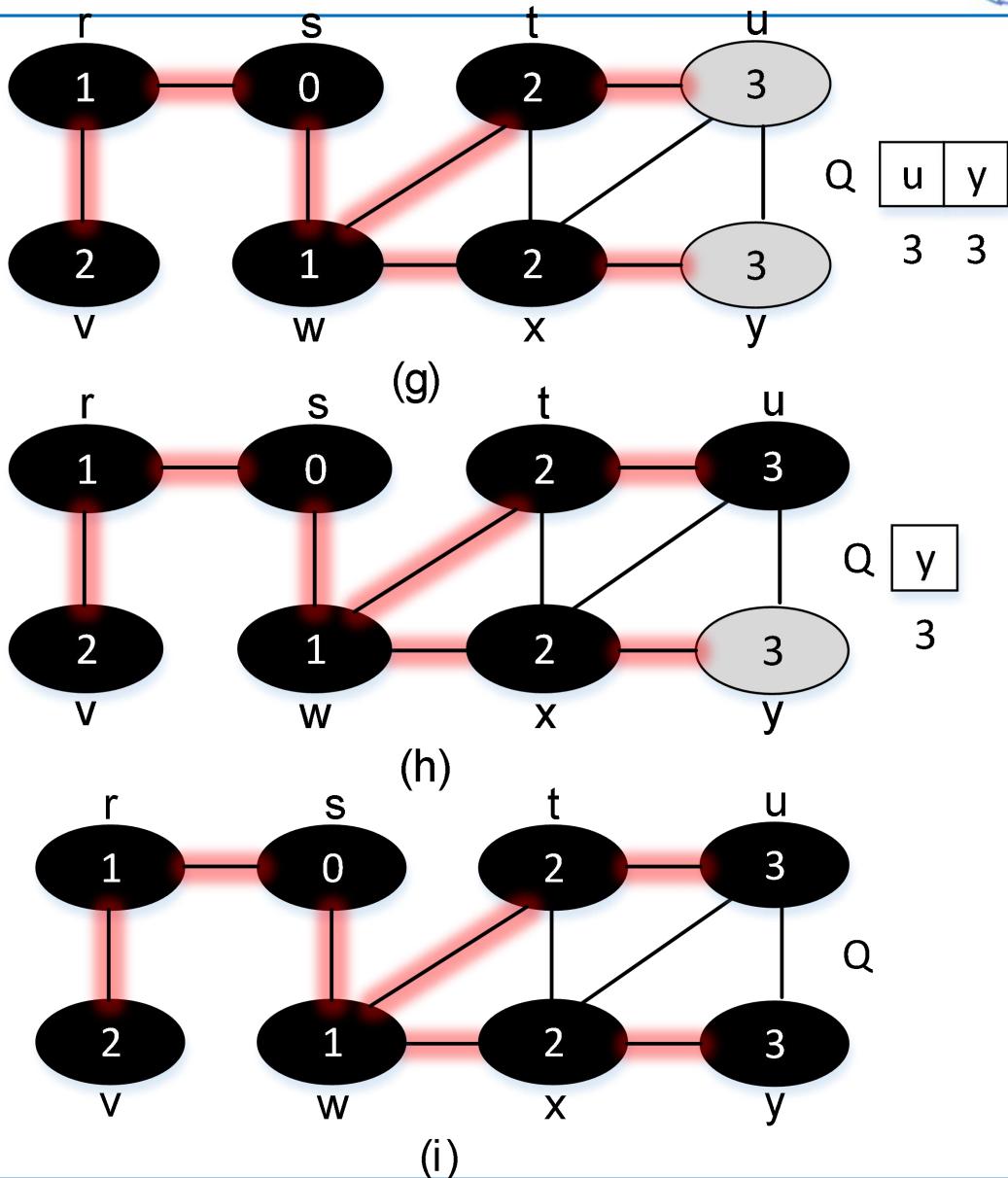


# 2 广度优先搜索-BFS实现过程

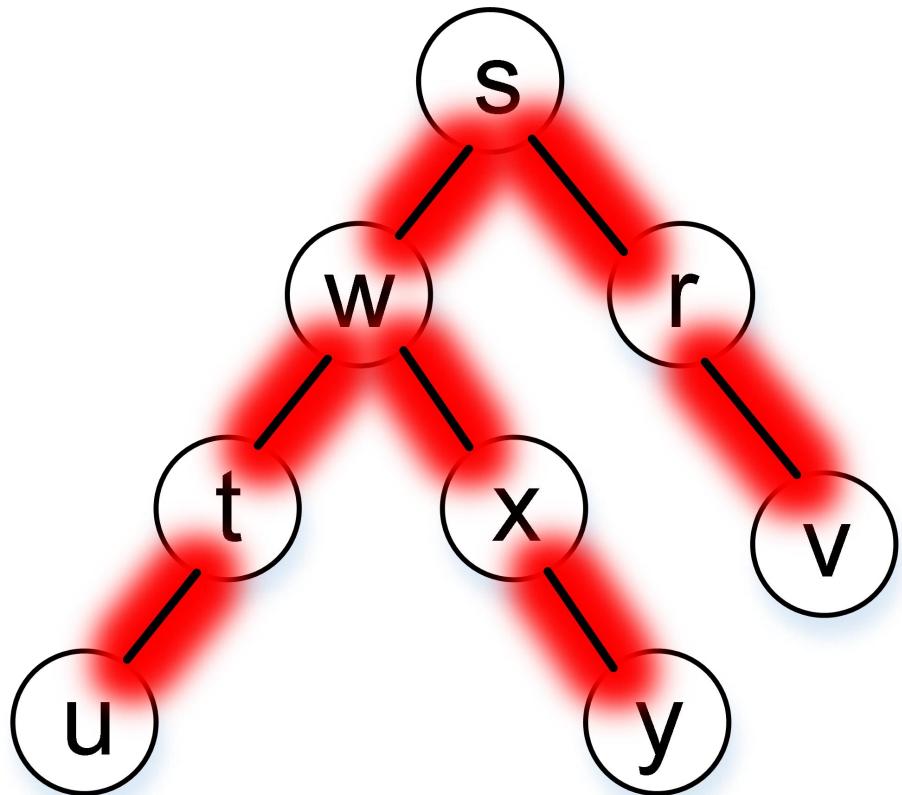
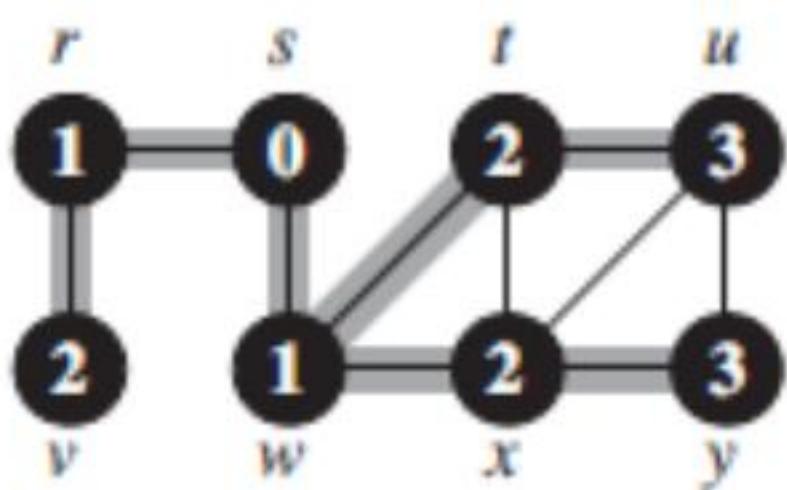
BFS( $G, s$ )

```

1 for each vertex  $u \in G.V - \{s\}$ 
2    $u.color = \text{WHITE}$ 
3    $u.d = \infty$ 
4    $u.\pi = \text{NIL}$ 
5  $s.color = \text{GRAY}$ 
6  $s.d = 0$ 
7  $s.\pi = \text{NIL}$ 
8  $Q = \emptyset$ 
9 ENQUEUE( $Q, s$ )
10 while  $Q \neq \emptyset$ 
11    $u = \text{DEQUEUE}(Q)$ 
12   for each  $v \in G.Adj[u]$ 
13     if  $v.color == \text{WHITE}$ 
14        $v.color = \text{GRAY}$ 
15        $v.d = u.d + 1$ 
16        $v.\pi = u$ 
17       ENQUEUE( $Q, v$ )
18    $u.color = \text{BLACK}$ 
```



# 2广度优先搜索-广度优先树



# 2广度优先搜索-算法时间复杂度分析



BFS( $G, s$ )

```
1  for each vertex  $u \in G.V - \{s\}$ 
2       $u.color = \text{WHITE}$ 
3       $u.d = \infty$ 
4       $u.\pi = \text{NIL}$ 
5       $s.color = \text{GRAY}$ 
6       $s.d = 0$ 
7       $s.\pi = \text{NIL}$ 
8       $Q = \emptyset$ 
9      ENQUEUE( $Q, s$ )
10     while  $Q \neq \emptyset$ 
11          $u = \text{DEQUEUE}(Q)$ 
12         for each  $v \in G.Adj[u]$ 
13             if  $v.color == \text{WHITE}$ 
14                  $v.color = \text{GRAY}$ 
15                  $v.d = u.d + 1$ 
16                  $v.\pi = u$ 
17                 ENQUEUE( $Q, v$ )
18          $u.color = \text{BLACK}$ 
```

每个点在queue中最多出现一次

- 时间代价为 $O(|V|)$

邻接表Adj也只会被扫描一次

- 时间代价为

$$\sum_{v \in V} |Adj[v]| = O(|E|)$$

总时间代价为 $O(|V|+|E|)$

## 2广度优先搜索-最短路径 $\delta(s,v)$



可以证明，对于一个图 $G=(V,E)$ ，广度优先搜索算法可以得到从已知源结点 $s \in V$ 到每个可达结点的距离

### 引理1

设 $G=(V,E)$ 是一个有向图或无向图， $s \in V$ 为 $G$ 的任意一个结点，则对任意边 $(u,v) \in E$ ,  $\delta(s,v) \leq \delta(s,u)+1$

证明: 如果从顶点 $s$ 可达顶点 $u$ ，则从 $s$ 也可达 $v$ 。在这种情况下从 $s$ 到 $v$ 的最短路径不可能比从 $s$ 到 $u$ 的最短路径加上边 $(u,v)$ 更长，因此不等式成立；如果从 $s$ 不可达顶点 $u$ ，则 $\delta(s,v)=\infty$ ，不等式仍然成立。

# 2广度优先搜索-最短路径 $\delta(s,v)$



下面我们首先证明 $v.d$ 是 $\delta(s,v)$ 的上界

## 引理2

设 $G=(V,E)$ 是一个有向或无向图，并假设计算法BFS从 $G$ 中一已知源结点 $s \in V$ 开始执行，在执行终止时，对每个顶点 $v \in V$ ，变量 $v.d$ 的值满足： $v.d \geq \delta(s,v)$ 。

证明：利用数学归纳法，对 $s$ 到 $v$ 的BFS路径长度 $L$ 进行归纳

$L=0$ 时，只有 $s$ 这个点 $L=0$ ，首先被放入queue，此时 $v.d=0$ ；

$L \leq n$ 时，假设所有点 $v$ 都成立；当 $L=n+1$ 时，假设 $s$ 到 $v$ 路径为 $s e_1 v_1 e_2 \dots e_n v_n e_{n+1} v_{n+1}$ ，根据归纳假设 $v_n$ 加入queue的时候 $v_n.d$ 等于或大于 $n$ ，同时边 $e_{n+1}$ 的存在保证了 $v_{n+1}$ 加入的时候 $v_{n+1}.d = v_n.d + 1 \geq n + 1$

## 2广度优先搜索-最短路径 $\delta(s,v)$



本引理的目的在于证明，队列中最多只有两个不同值

### 引理3

假设过程BFS在图 $G=(V,E)$ 之上的执行过程中，队列 $Q$ 包含如下结点 $\langle v_1, v_2, \dots, v_r \rangle$ ，其中 $v_1$ 是队列 $Q$ 的头， $v_r$ 是队列的尾。那么 $v_r.d \leq v_1.d + 1$ 且 $v_i.d \leq v_{i+1}.d$ ,  $i=1, 2, \dots, r-1$ 。

证明：

证明过程是对队列操作的次数进行归纳。初始时，队列仅包含顶点 $s$ ，引理自然正确。

下面进行归纳，我们必须证明在压入和弹出一个顶点后引理仍然成立。如果队列的头 $v_1$ 被弹出队列，新的队头为 $v_2$ (如果此时队列为空，引理无疑成立)，所以有 $v_r.d \leq v_1.d + 1 \leq v_2.d + 1$ ，余下的不等式依然成立，因此 $v_2$ 为队头时引理成立

## 2广度优先搜索-最短路径 $\delta(s,v)$



上面引理证明了队列中 $d$ 的值在不断增加，同时算法伪代码显示该算法每个点只会入队列一次

### 引理4

设 $G=(V,E)$ 是一个有向图或无向图，并假设过程BFS从 $G$ 上某顶点 $s \in V$ 开始执行，则在执行过程中，BFS可以发现源结点 $s$ 可达的每一个结点 $v \in V$ ，在运行终止时，对任意 $v \in V$ ,  $v.d = \delta(s,v)$ 。此外，对任意从 $s$ 可达的节点 $v \neq s$ ，从 $s$ 到 $v$ 的最短路径之一是从 $s$ 到 $v$ . $\pi$ 的最短路径再加上边 $(v, \pi, v)$ 。



## 2广度优先搜索-最短路径 $\delta(s,v)$

引理4证明：

设反证，假设存在 $v$ ,  $v.d$ 不等于 $\delta(s,v)$ , 且 $\delta(s,v)$ 最小。显然,  $v$ 不是 $s$

根据引理2,  $v.d \geq \delta(s,v)$ , 于是 $v.d$ 大于 $\delta(s,v)$ , 注意 $v$ 必然可达, 否则  
 $\delta(s,v)$ 为无穷了, 那么 $v.d$ 不可能更大了

假设 $u$ 是 $s$ 到 $v$ 最短路径上 $v$ 前面一个点, 所以 $\delta(s,v) = \delta(s,u) + 1$ 。同时,  
 $\delta(s,v)$ 最小, 所以 $\delta(s,u) = u.d$ 。于是, 我们有

$$v.d > \delta(s,v) = \delta(s,u) + 1 = u.d + 1.$$

这个不可能的



# 提纲

1 图的基本概念

2 广度优先搜索

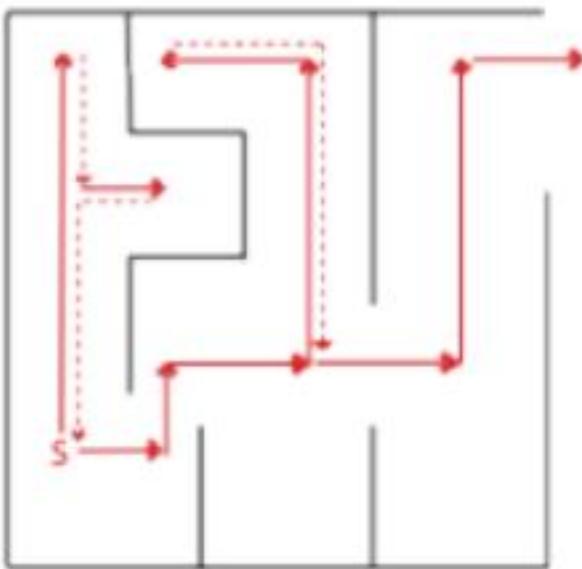
3 深度优先搜索

4 深度优先搜索应用



### 3深度优先搜索

深度优先搜索（Depth First Search），从初始点开始，对每一个可能的分支路径深入到不能再深入为止，而且每个节点只能访问一次。





### 3深度优先搜索

- 令  $G=(V, E)$  是一个有向图或无向图，搜索时沿着树的深度遍历树的结点，尽可能深的搜索树的分支。
- 当结点  $u$  的所有边都已被搜索过，搜索将回溯到发现结点  $u$  的那条边的起始结点。这一过程一直进行到已发现从源结点可达的所有结点为止。
- 如果还存在未被发现的结点，则选择其中一个作为源结点并重复以上过程，整个进程反复进行直到所有结点都被访问为止。



# 算法伪代码

DFS(V, Adj, s)

    level = {s:0}; parent = {s: None};

    stack=[s];

    while stack

        cur = stack[len(stack)-1]; tag = 0;

        for v in Adj[cur]

            if v not in level #not yet seen

                level[v] = level[cur] + 1; parent[v] = cur;

                stack.append(v); tag = 1

            if tag == 0

                stack.pop();

# 3深度优先搜索-DFS实现过程：递归



$\text{DFS}(G)$

```

1 for each vertex  $u \in G.V$ 
2    $u.\text{color} = \text{WHITE}$ 
3    $u.\pi = \text{NIL}$ 
4    $\text{time} = 0$ 
5 for each vertex  $u \in G.V$ 
6   if  $u.\text{color} == \text{WHITE}$ 
7      $\text{DFS-VISIT}(G, u)$ 

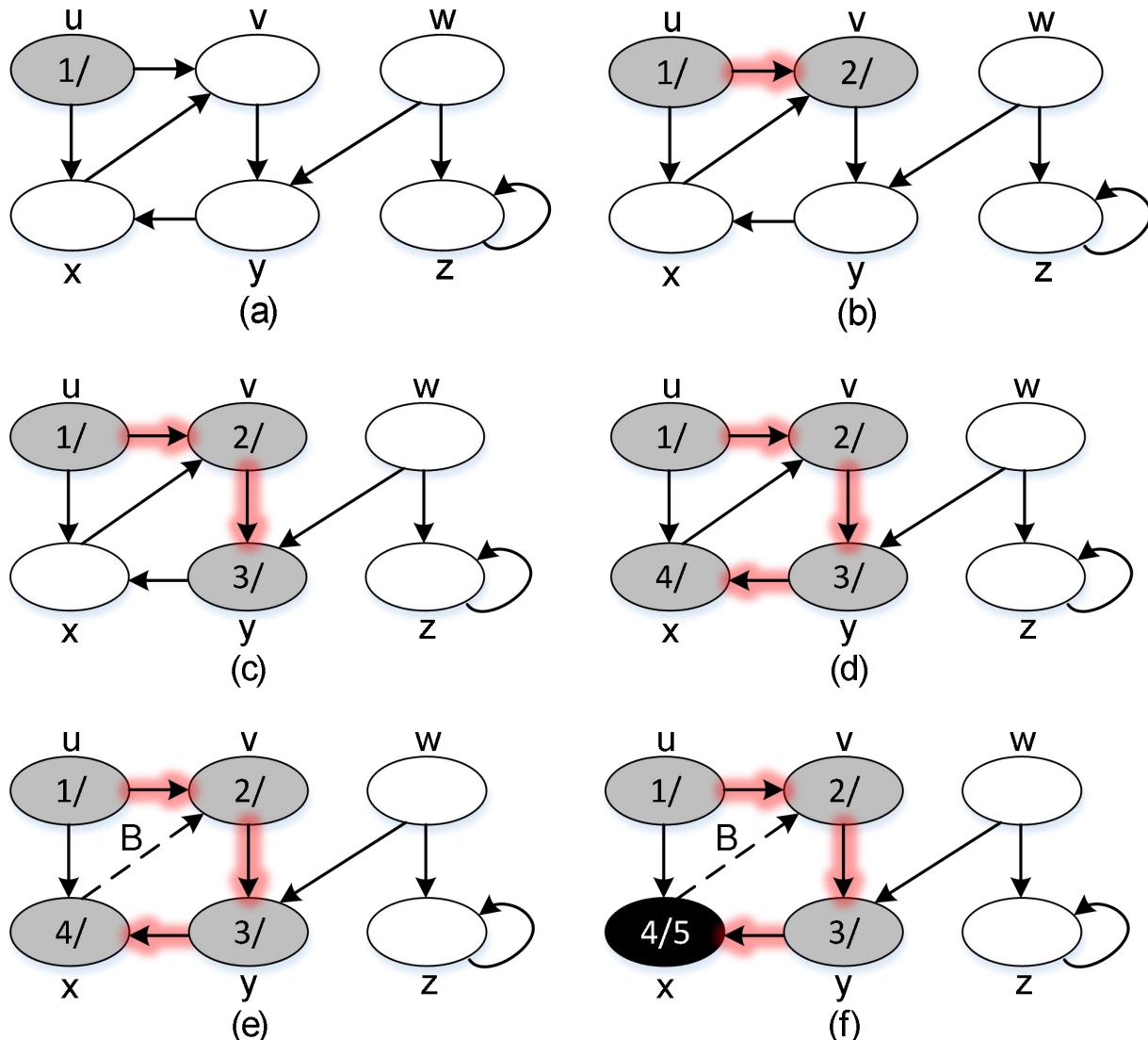
```

$\text{DFS-VISIT}(G, u)$

```

1  $\text{time} = \text{time} + 1$ 
2  $u.d = \text{time}$ 
3  $u.\text{color} = \text{GRAY}$ 
4 for each  $v \in G.\text{Adj}[u]$ 
5   if  $v.\text{color} == \text{WHITE}$ 
6      $v.\pi = u$ 
7      $\text{DFS-VISIT}(G, v)$ 
8    $u.\text{color} = \text{BLACK}$ 
9    $\text{time} = \text{time} + 1$ 
10  $u.f = \text{time}$ 

```



# 3深度优先搜索-DFS实现过程：递归



DFS( $G$ )

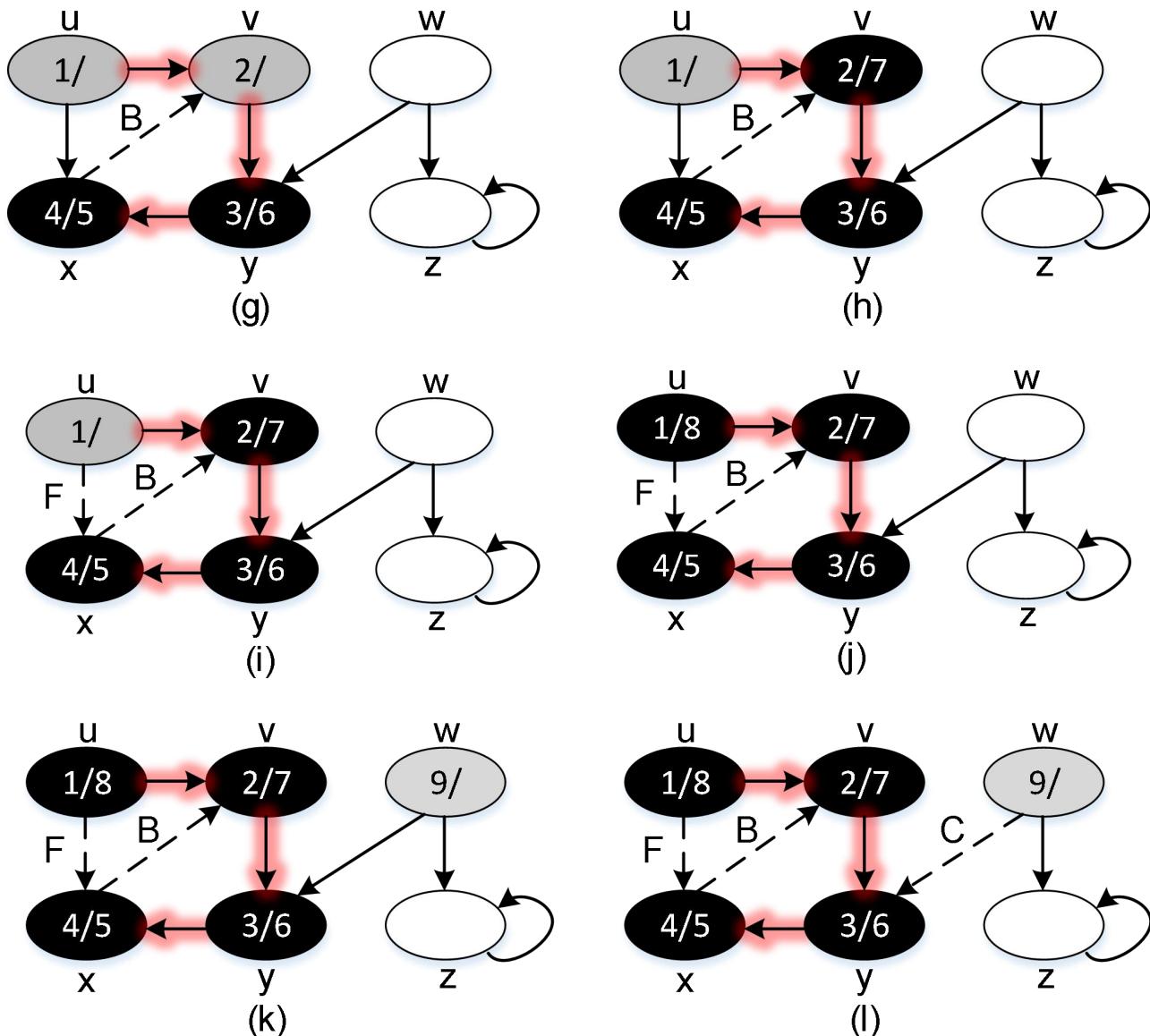
```

1 for each vertex  $u \in G.V$ 
2    $u.color = \text{WHITE}$ 
3    $u.\pi = \text{NIL}$ 
4    $time = 0$ 
5 for each vertex  $u \in G.V$ 
6   if  $u.color == \text{WHITE}$ 
7     DFS-VISIT( $G, u$ )
  
```

DFS-VISIT( $G, u$ )

```

1  $time = time + 1$ 
2  $u.d = time$ 
3  $u.color = \text{GRAY}$ 
4 for each  $v \in G.Adj[u]$ 
5   if  $v.color == \text{WHITE}$ 
6      $v.\pi = u$ 
7     DFS-VISIT( $G, v$ )
8    $u.color = \text{BLACK}$ 
9    $time = time + 1$ 
10   $u.f = time$ 
  
```



# 3深度优先搜索-DFS实现过程：递归



$\text{DFS}(G)$

```

1 for each vertex  $u \in G.V$ 
2    $u.\text{color} = \text{WHITE}$ 
3    $u.\pi = \text{NIL}$ 
4    $\text{time} = 0$ 
5 for each vertex  $u \in G.V$ 
6   if  $u.\text{color} == \text{WHITE}$ 
7      $\text{DFS-VISIT}(G, u)$ 

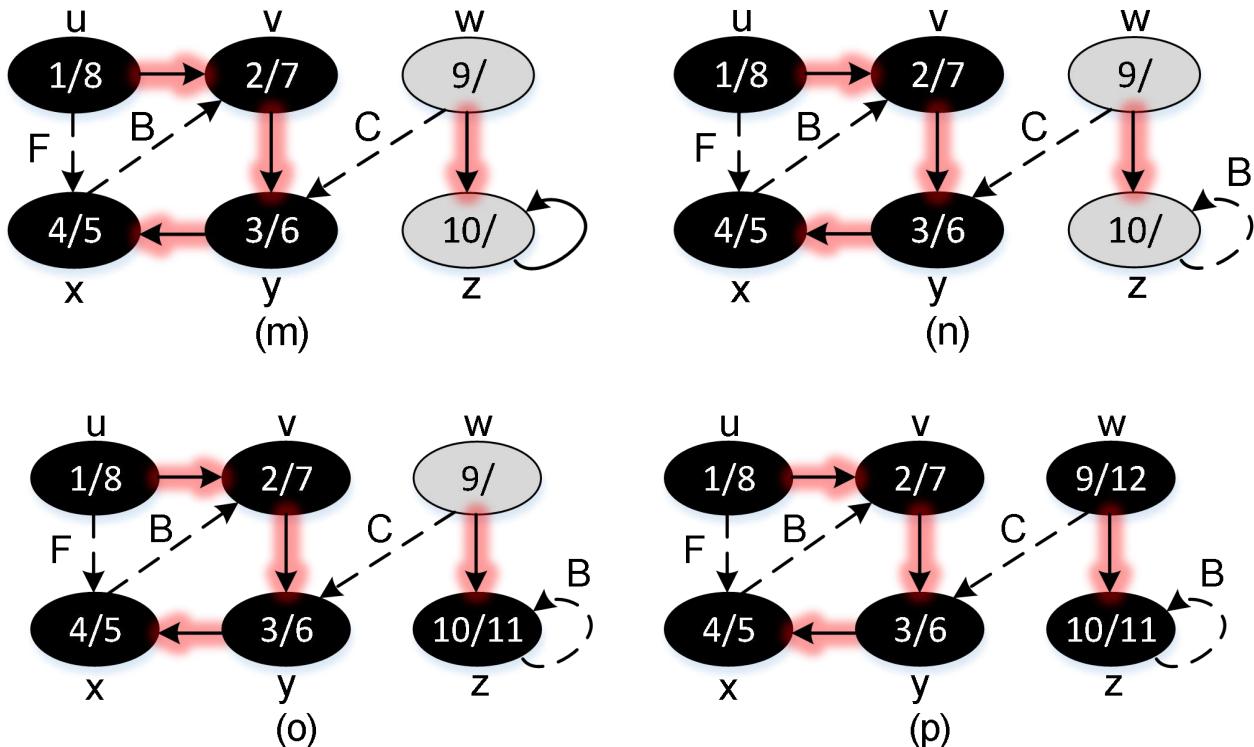
```

$\text{DFS-VISIT}(G, u)$

```

1  $\text{time} = \text{time} + 1$ 
2  $u.d = \text{time}$ 
3  $u.\text{color} = \text{GRAY}$ 
4 for each  $v \in G.\text{Adj}[u]$ 
5   if  $v.\text{color} == \text{WHITE}$ 
6      $v.\pi = u$ 
7      $\text{DFS-VISIT}(G, v)$ 
8    $u.\text{color} = \text{BLACK}$ 
9    $\text{time} = \text{time} + 1$ 
10  $u.f = \text{time}$ 

```



# 3深度优先搜索-DFS时间复杂度分析



DFS( $G$ )

```
1 for each vertex  $u \in G.V$ 
2    $u.color = \text{WHITE}$ 
3    $u.\pi = \text{NIL}$ 
4    $time = 0$ 
5   for each vertex  $u \in G.V$ 
6     if  $u.color == \text{WHITE}$ 
7       DFS-VISIT( $G, u$ )
```

第1-3行和5-7行的循环占用时间为 $O(V)$

DFS\_Visit( $v$ )第4-7行的循环要执行 $|Adj[v]|$ 次

$$\sum_{v \in V} |Adj[v]| = \Theta(E)$$

DFS-VISIT( $G, u$ )

```
1  $time = time + 1$ 
2  $u.d = time$ 
3  $u.color = \text{GRAY}$ 
4 for each  $v \in G.Adj[u]$ 
5   if  $v.color == \text{WHITE}$ 
6      $v.\pi = u$ 
7     DFS-VISIT( $G, v$ )
8    $u.color = \text{BLACK}$ 
9    $time = time + 1$ 
10  $u.f = time$ 
```

DFS\_Visit中第4-7行语句占用的整个时间应为 $\Theta(E)$

DFS的运行时间为 $\Theta(V+E)$



# 3深度优先搜索-DFS实现：栈

伪代码

1. 栈初始化；
2. 输出起始顶点；起始顶点改为“已访问”标志；将起始顶点进栈；
3. 重复下列操作直到栈为空：
  - 3.1 取栈顶元素顶点；（注意不出栈）
  - 3.2 栈顶元素顶点存在未被访问过的邻接点 w，则
    - 3.2.1 输出顶点 w
    - 3.2.2 将顶点 w 改为“已访问”标志；
    - 3.2.3 将顶点 w 进栈；
  - 3.3 否则，当前顶点退栈；



# BFS vs DFS

## 相似点

都维护了一个任务列表来  
不断地插入和推出未处理  
的点

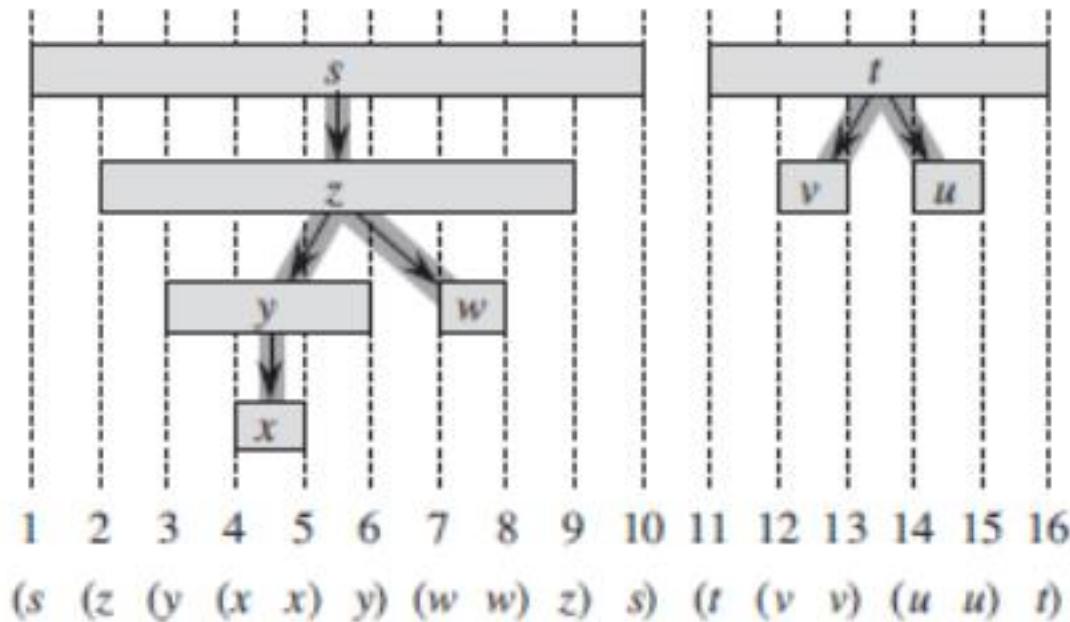
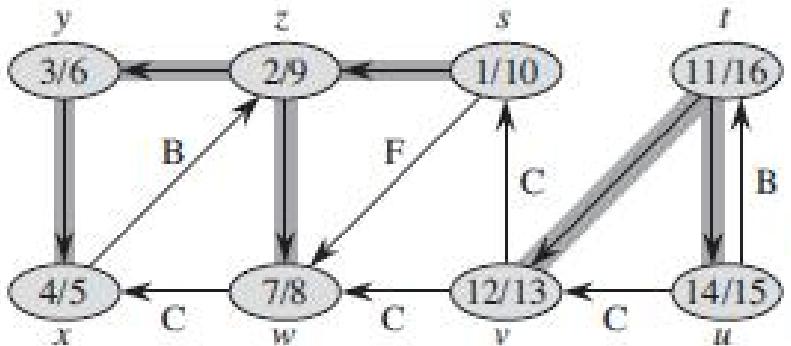
## 不同点

BFS使用队列； DFS使用栈  
  
BFS按层遍历图； DFS贪心  
地搜索



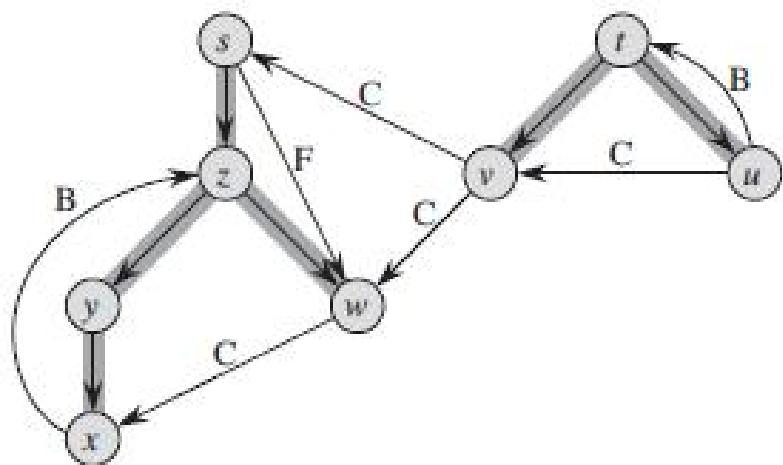
# 3深度优先搜索-性质

从不同的点出发或者沿着不同点深优，会得到不同深度优先搜索森林。



每个连通分支对应一棵树。

强连通，只有一个连通分支，  
深度优先搜索森林中只有一  
棵树，弱连通则多棵树。





### 3深度优先搜索-性质

注意：有向图中，沿不同的点出发或者沿着不同点深优，会得到不同深度优先搜索森林。如果是强连通，则都只有一棵树。如果是弱连通，则不同的点出发或者沿着不同点深优，得到深度优先搜索森林中树的个数可能不同。

注意：无向图中，沿不同的点出发或者沿着不同点深优，会得到不同深度优先搜索森林。

练习：无向图中，沿不同的点出发或者沿着不同点深优，会得到不同深度优先搜索森林，那森林中含有树的数量相同吗？



# 3深度优先搜索-性质

## 定理1 括号定理

在对有向图或无向图 $G=(V, E)$ 的任何深度优先搜索中，对于图中任意两结点 $u$ 和 $v$ ，下述三个条件中有且仅有一条成立：

- 区间 $[u.d, u.f]$ 和区间 $[v.d, v.f]$ 完全分离，在深度优先森林中，结点 $u$ 不是结点 $v$ 的后代，结点 $v$ 也不是 $u$ 的后代；
- 区间 $[u.d, u.f]$ 完全包含于区间 $[v.d, v.f]$ 中，在深度优先树中 $u$ 是 $v$ 的后代；
- 区间 $[v.d, v.f]$ 完全包含于区间 $[u.d, u.f]$ 中，在深度优先树中 $u$ 是 $v$ 的祖先。

证明：先讨论 $u.d < v.d$ 的情形，根据 $v.d$ 是否小于 $u.d$ 又可分为两种情况：第一种情况，若 $v.d < u.f$ ，这样 $v$ 已被发现时 $u$ 结点依然是灰色，这就说明 $v$ 是 $u$ 的后裔，再者，因为结点 $v$ 比 $u$ 发现得较晚，所以在搜索返回结点 $u$ 并完成之前，所有从 $v$ 出发的边都已被探寻并已完成，所以在这种条件下区间 $[v.d, v.f]$ 必然完全包含于区间 $[u.d, u.f]$ 。第二种情况，若 $u.f < v.d$ ，则根据不等式(1)，区间 $[u.d, u.f]$ 和区间 $[v.d, v.f]$ 必然是分离的。对于 $v.d < u.d$ 的情形类似可证，只要把上述证明中 $u$ 和 $v$ 对调即可。



# 3深度优先搜索-性质

## 推论1 后代区间的嵌套

在有向或无向图G的深度优先森林中，结点v是结点u的后裔当且仅当 $u.d < v.d < v.f < u.f$ 。



# 3深度优先搜索-性质

## 定理2 白色路径定理

在一个有向或无向图 $G=(V,E)$ 的深度优先森林中，结点 $v$ 是结点 $u$ 的后代当且仅当在搜索发现 $u$ 的时刻 $u.d$ ，从结点 $u$ 出发经一条仅由白色结点组成的路径可达 $v$ 。

证明：

→：假设 $v$ 是 $u$ 的后裔， $w$ 是深度优先树中 $u$ 和 $v$ 之间的通路上的任意结点，则 $w$ 必然是 $u$ 的后裔，由推论1可知 $u.d < w.d$ ，因此在时刻 $u.d$ ， $w$ 应为白色。

←：设在时刻 $u.d$ ，从 $u$ 到 $v$ 有一条仅由白色结点组成的通路，但在深度优先树中 $v$ 还没有成为 $u$ 的后裔。不失一般性，我们假定该通路上的其他顶点都是 $u$ 的后裔(否则可设 $v$ 是该通路中最接近 $u$ 的结点，且不为 $u$ 的后裔)，设 $w$ 为该通路上 $v$ 的祖先，使 $w$ 是 $u$ 的后裔(实际上 $w$ 和 $u$ 可以是同一个结点)。根据推论1得 $w.f \leq u.f$ ，因为 $v \in \text{Adj}[w]$ ，对 $\text{DFS\_Visit}(w)$ 的调用保证完成 $w$ 之前先完成 $v$ ，因此 $v.f < w.f \leq u.f$ 。因为在时刻 $u.d$ 结点 $v$ 为白色，所以有 $u.d < v.d$ 。由推论1可知在深度优先树中 $v$ 必然是 $u$ 的后裔。(证毕)

# 提纲



1 图的基本概念

2 广度优先搜索

3 深度优先搜索

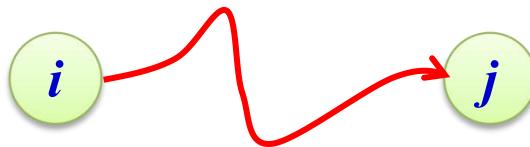
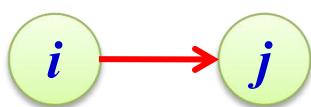
4 深度优先搜索应用



## 4.1 拓扑排序

设  $G=(V, E)$  是一个具有  $n$  个顶点的有向无环图， $V$  中顶点序列  $v_1, v_2, \dots, v_n$  称为一个拓扑序列，当且仅当该顶点序列满足下列条件：

若  $\langle i, j \rangle$  是图中的边（或从顶点  $i \Rightarrow j$  有一条路径）：



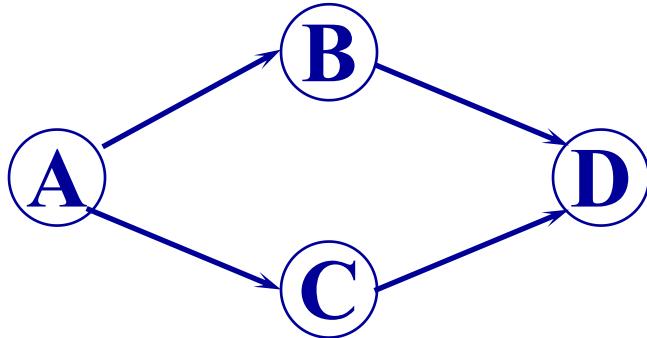
则在拓扑序列中顶点  $i$  必须排在顶点  $j$  之前。

在一个有向图中找一个拓扑序列的过程称为拓扑排序。

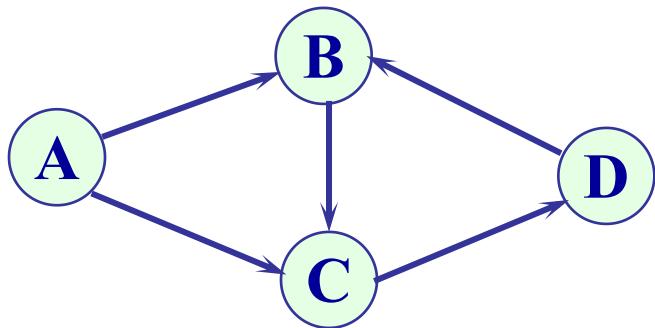


## 4.1 拓扑排序

有向无环图(DAG: Directed Acyclic Graph)



可求得拓扑有序序列: A B C D 或 A C B D

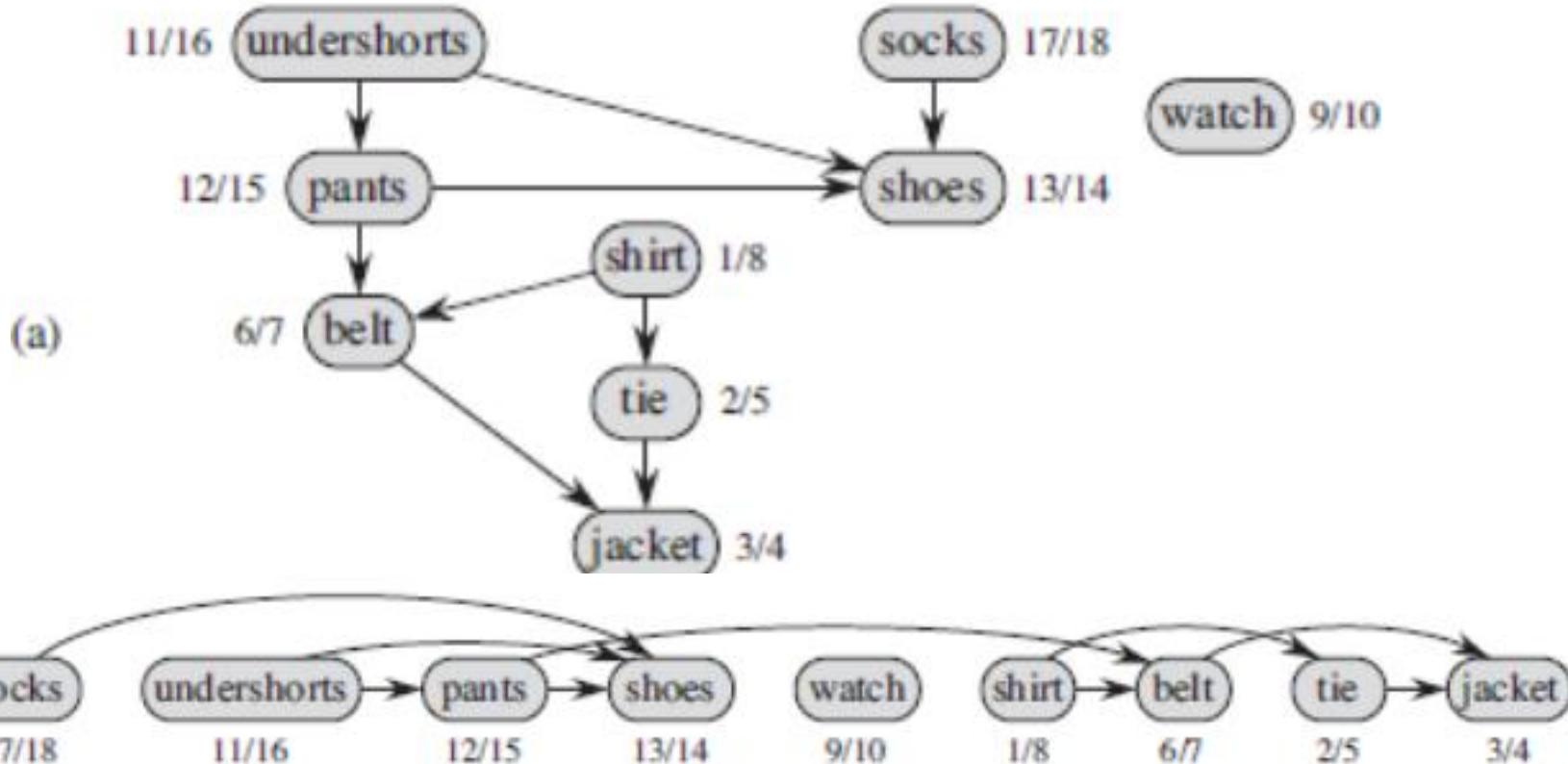


不能求得它的拓扑有序序列。因为存在回路 {B, C, D}



## 4.1 拓扑排序

对于有向无环图  $G=(V,E)$ ，其拓扑排序是  $G$  中所有结点的一种线性次序，该次序满足如下条件：如果  $G$  包含边  $(u, v)$ ，则结点  $u$  在拓扑排序中处于结点  $v$  的前面。





## 4.1 拓扑排序

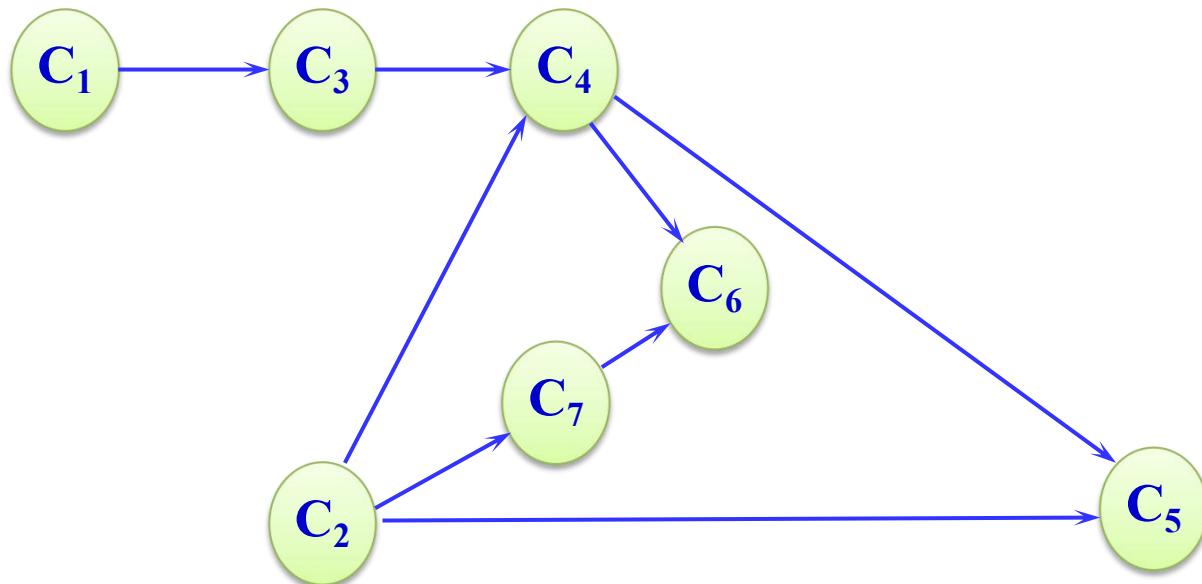
例如，计算机专业的学生必须完成一系列规定的课程才能毕业，假设这些课程的名称与相应代号有如下关系：

课程代号	课程名称	先修课程
C <sub>1</sub>	高等数学	无
C <sub>2</sub>	程序设计	无
C <sub>3</sub>	离散数学	C <sub>1</sub>
C <sub>4</sub>	数据结构	C <sub>2</sub> , C <sub>3</sub>
C <sub>5</sub>	编译原理	C <sub>2</sub> , C <sub>4</sub>
C <sub>6</sub>	操作系统	C <sub>4</sub> , C <sub>7</sub>
C <sub>7</sub>	计算机组成原理	C <sub>2</sub>



## 4.1 拓扑排序

课程之间的先后关系可用有向图表示：



可以这样排课：

- $C_1 - C_3 - C_2 - C_4 - C_7 - C_6 - C_5$
- $C_2 - C_7 - C_1 - C_3 - C_4 - C_5 - C_6$

第1学期

第2学期



## 4.1 拓扑排序

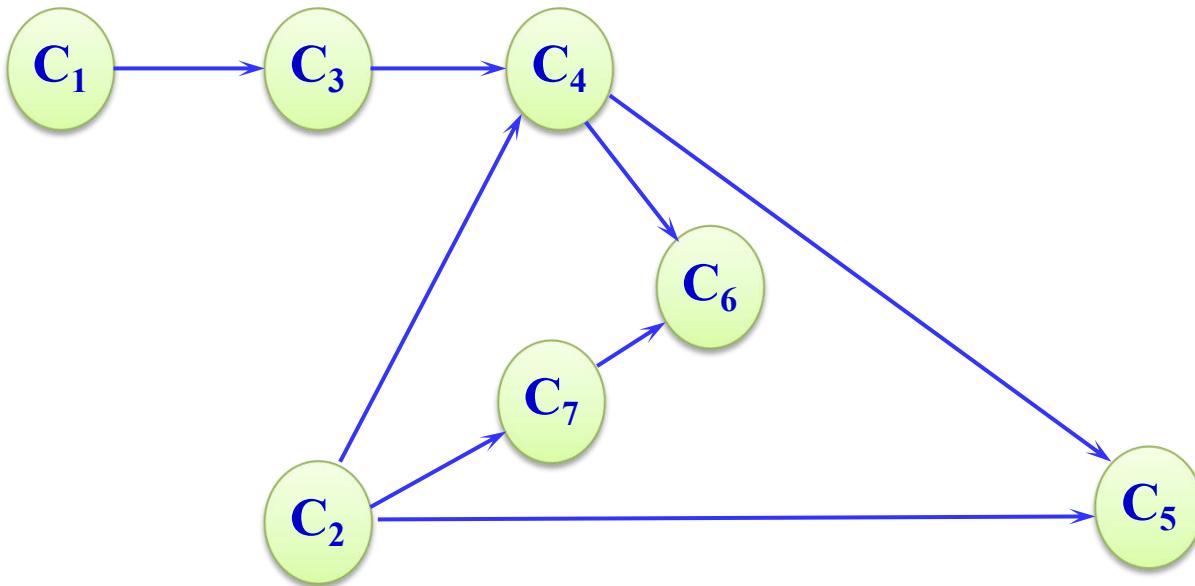
### 拓扑排序方法1：根据结点入度

- (1) 从有向图中选择一个没有前驱（即入度为0）的顶点并且输出它。
- (2) 从图中删去该顶点，并且删去从该顶点发出的全部有向边。
- (3) 重复上述两步，直到剩余的图中不再存在没有前驱的顶点为止。



# 4.1 拓扑排序

## 拓扑排序方法1演示



产生一个拓扑序列：

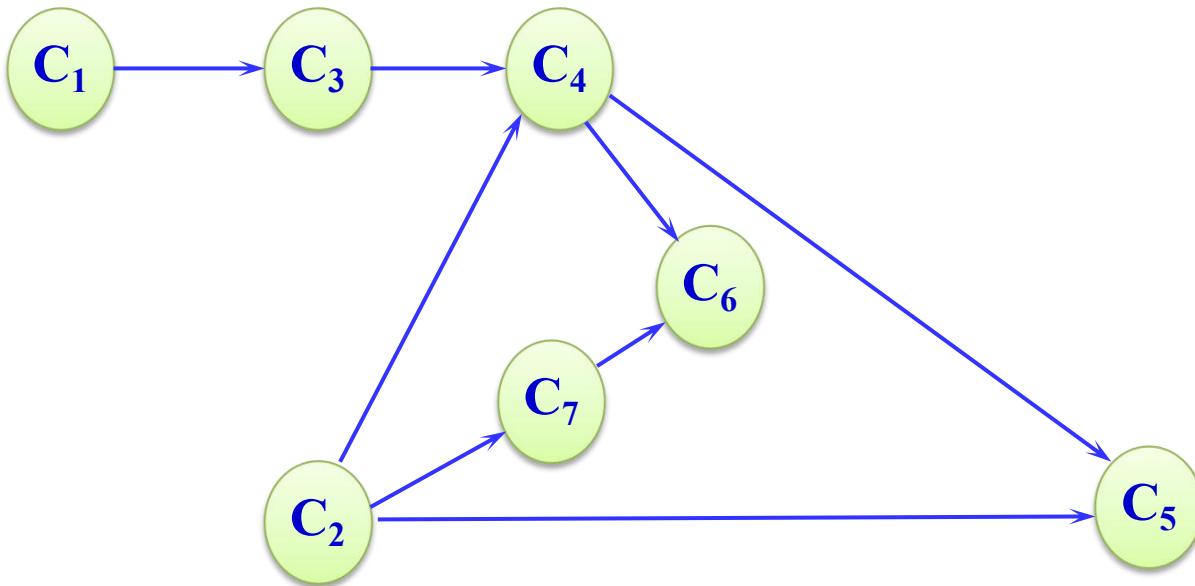
C<sub>1</sub>      C<sub>2</sub>      C<sub>3</sub>      C<sub>4</sub>      C<sub>5</sub>      C<sub>7</sub>      C<sub>6</sub>

排序完成



# 4.1 拓扑排序

## 拓扑排序方法1演示



产生一个拓扑序列：

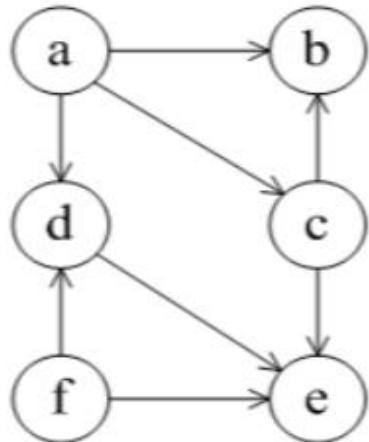
C<sub>1</sub>      C<sub>2</sub>      C<sub>3</sub>      C<sub>4</sub>      C<sub>5</sub>      C<sub>7</sub>      C<sub>6</sub>

排序完成

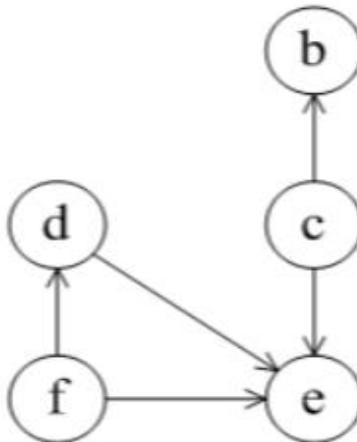


# 4.1 拓扑排序

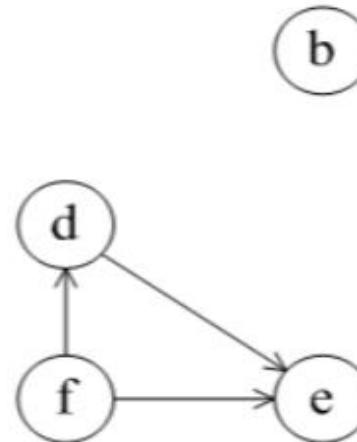
## 拓扑排序方法1 演示



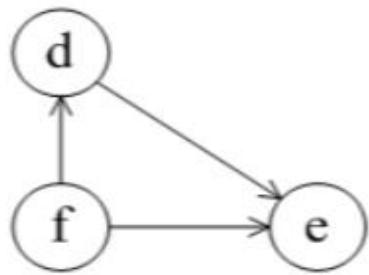
(a)图



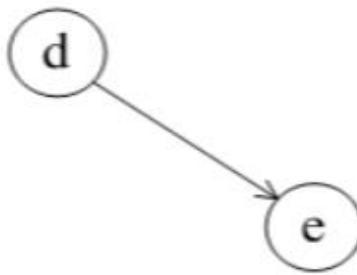
(b)输出a



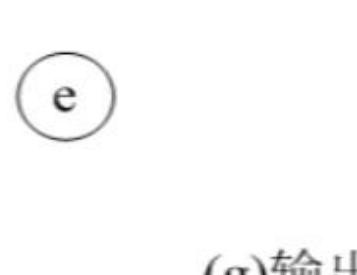
(c)输出c



(d)输出b



(e)输出f



(f)输出d (g)输出  
e

拓扑排序序列： acbfde





# 4.1 拓扑排序

## 拓扑排序方法2：基于DFS

$\text{DFS}(G)$

```
1 for each vertex  $u \in G.V$ 
2    $u.\text{color} = \text{WHITE}$ 
3    $u.\pi = \text{NIL}$ 
4    $time = 0$ 
5 for each vertex  $u \in G.V$ 
6   if  $u.\text{color} == \text{WHITE}$ 
7      $\text{DFS-VISIT}(G, u)$ 
```

$\text{DFS-VISIT}(G, u)$

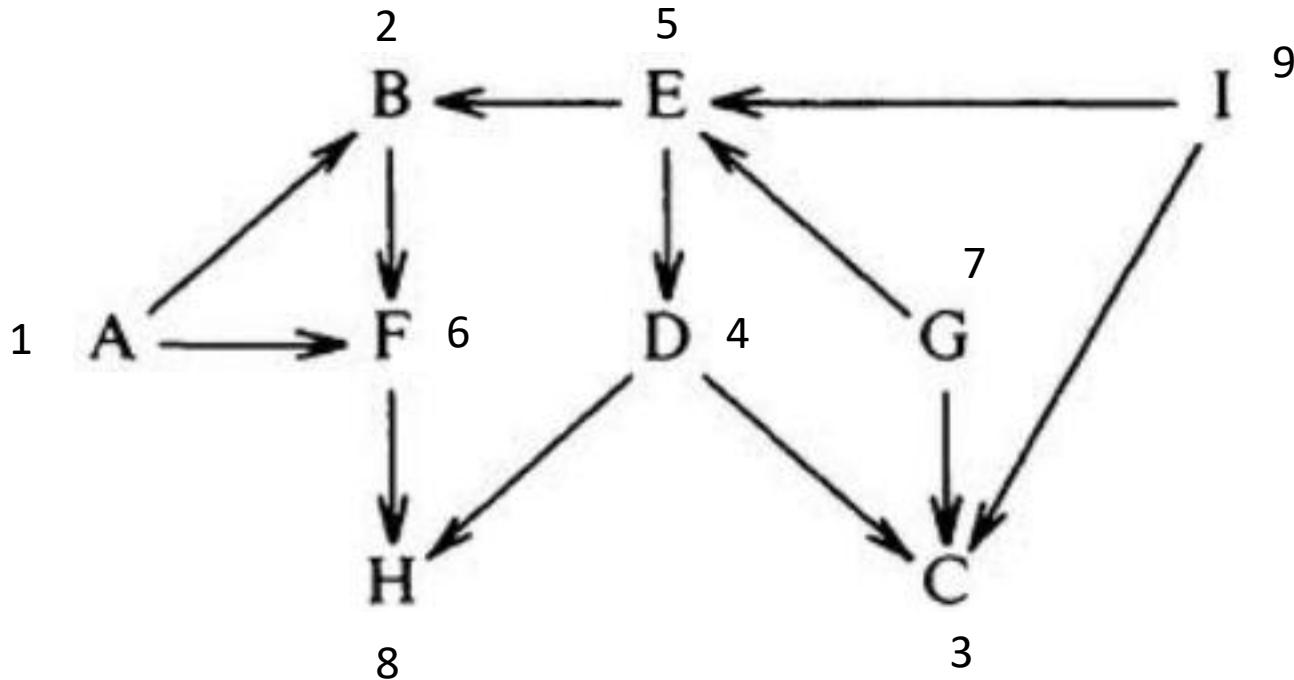
```
1  $time = time + 1$ 
2  $u.d = time$ 
3  $u.\text{color} = \text{GRAY}$ 
4 for each  $v \in G.\text{Adj}[u]$ 
5   if  $v.\text{color} == \text{WHITE}$ 
6      $v.\pi = u$ 
7      $\text{DFS-VISIT}(G, v)$ 
8    $v.\text{color} = \text{BLACK}$ 
9    $time = time + 1$ 
10   $u.f = time$ 
```

$\text{TOPOLOGICAL-SORT}(G)$

- 1 call  $\text{DFS}(G)$  to compute finishing times  $v.f$  for each vertex  $v$
- 2 as each vertex is finished, insert it onto the front of a linked list
- 3 return the linked list of vertices

# 4.1 拓扑排序

## 拓扑排序方法2：基于DFS





## 4.1 拓扑排序

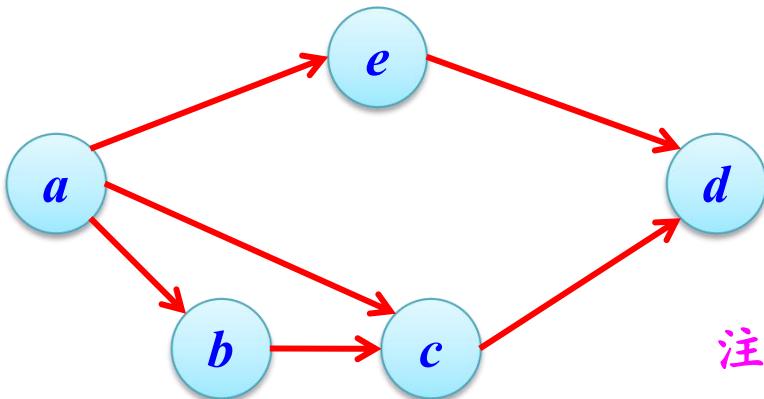
**【练习】** 对如图所示的图进行拓扑排序，可以得到不同的拓扑序列个数是\_\_\_\_\_。

A. 4

B. 3

C. 2

D. 1



注：2010年全国考研题

**解：**不同的拓扑序列有：*aebcd*、*abecd*、*abced*。答案为B。



## 4.2 XPath查询处理

Nicolas Bruno, Nick Koudas, Divesh Srivastava. Holistic twig joins: optimal XML pattern matching. SIGMOD Conference 2002: 310-321



## 4.2 XPath查询处理

XML 指可扩展标记语言（EXtensible Markup Language），用以表示数据以方便传输

XML 文档形成了一种树结构，它从“根部”开始，然后扩展到“枝叶”

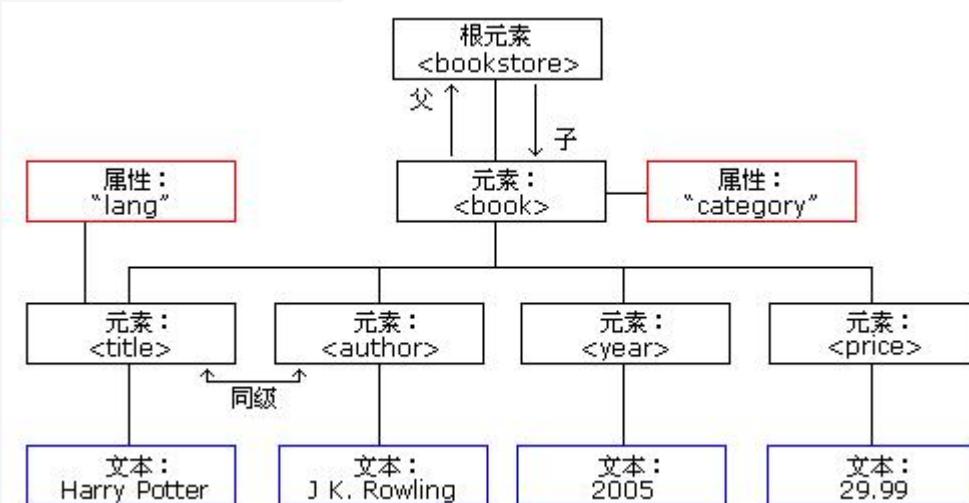
XML数据库就是多个XML 文档组成的一个数据库



## 4.2 XPath查询处理

### XML文档树示例

```
<bookstore>
  <book category="COOKING">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="CHILDREN">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="WEB">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```





## 4.2 XPath查询处理

### XPath 简介

XPath 是一门在 XML 文档中查找信息的语言。XPath 用于在 XML 文档中通过元素和属性进行导航。

Xpath查询: //a//b//c

➤Xpath查询的路径表示:

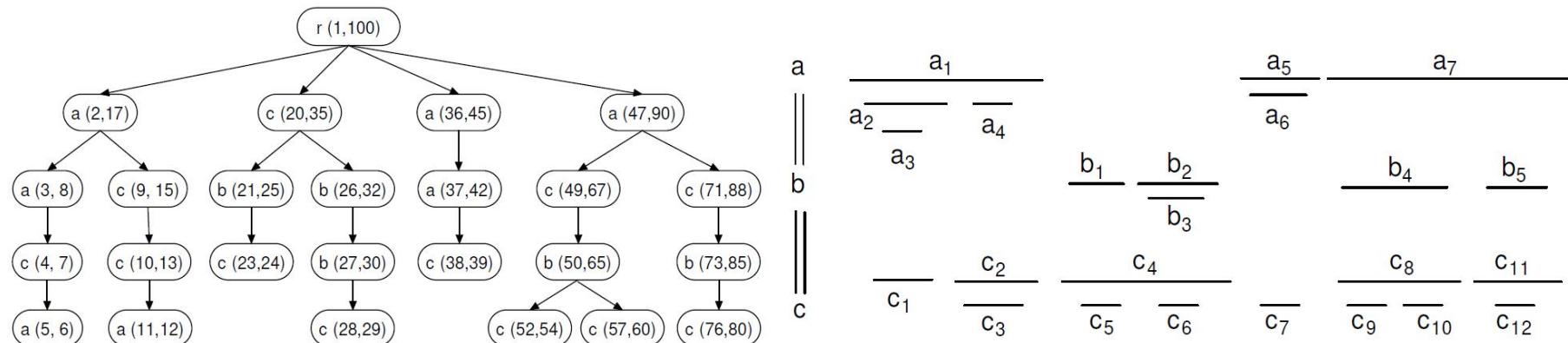
a  
||  
b  
||  
c



# 4.2 XPath查询处理

## DFS编码

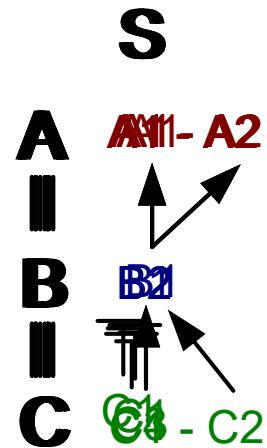
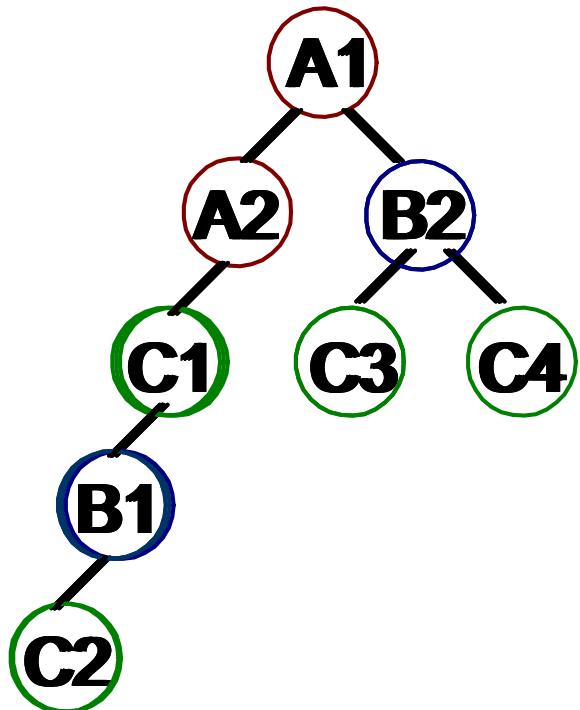
每个查询点的候选流（按照**发现时间**排序）、针对每个查询变量的栈





## 4.2 XPath查询处理

### PathStack示例



A1, B1, C2
A1, B1, C2
A2, B1, C2
A2, B1, C2
A2, B2, C3
A2, B2, C3
A2, B2, C4
A1, B2, C4

时间复杂度  $O(|\text{input}| + |\text{output}|)$