第八章 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

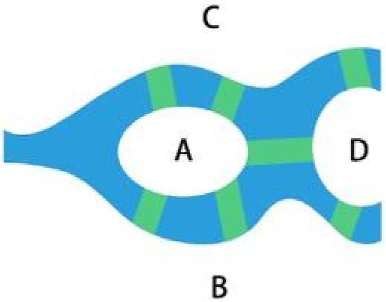
4 深度优先搜索应用

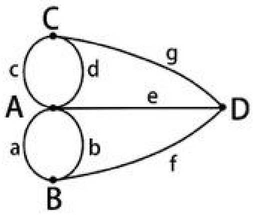
图论的起源

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

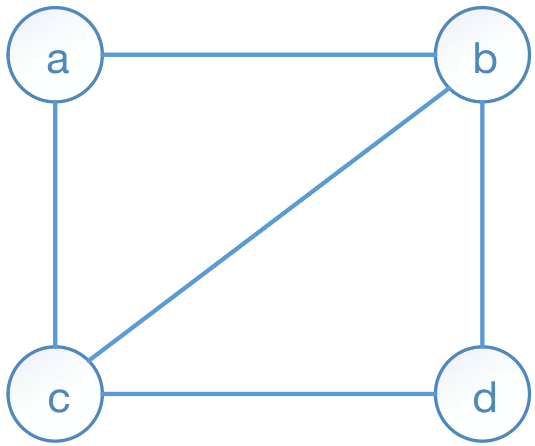
图的定义

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

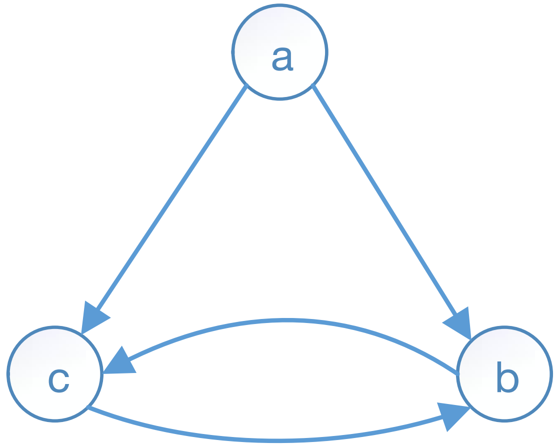
图可以表示成一个二元组$G = < V, E >$，其中V 是点集合，$E \subseteq V \times V$ 表示边集合，其中每条边由V 中两个点相连接所构成

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$\Gamma = v_0e_1v_1e_2 \dots e_lv_l$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（$1 \le r \le l$）

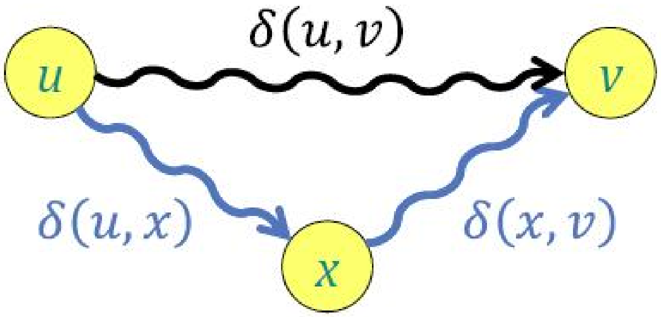
有向图中**路径**的定义与无向图中定义类似，只是要注意在有向图中通路和回路中边的方向的一致性，即v_{r-1} 必须是e_r 的起点且v_r 是e_r 的终点（$1 \le r \le l$）

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∞

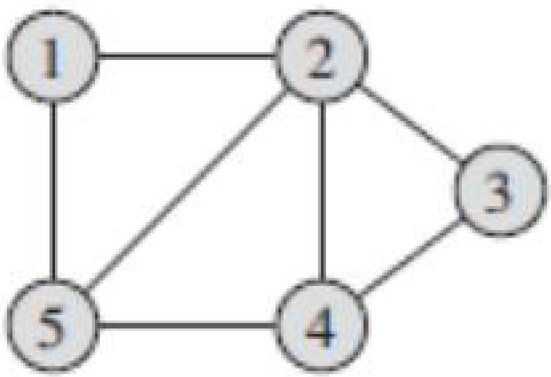
三角不等式

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

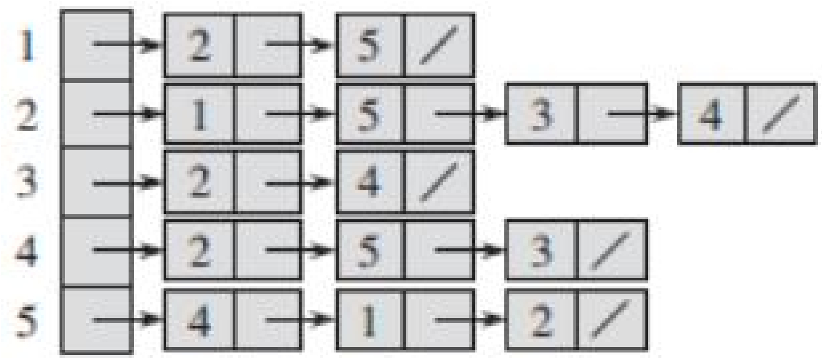


推论：对于图上任意三个点u、v和x且存在边(x,v)而言，$\delta(u,v) \le \delta(u,x)+1$

无向图的表示



(a)



(b)

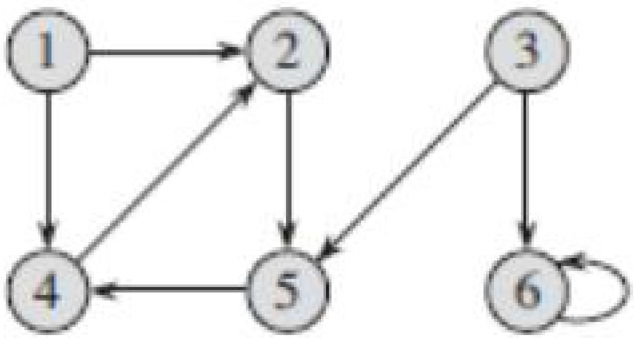
邻接表

	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

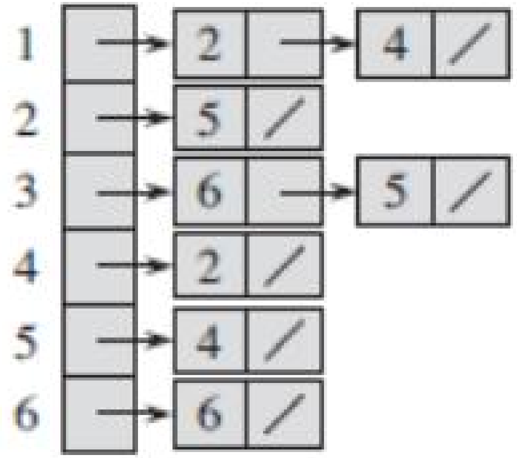
(c)

邻接矩阵

有向图的表示



(a)



(b)

邻接表

	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

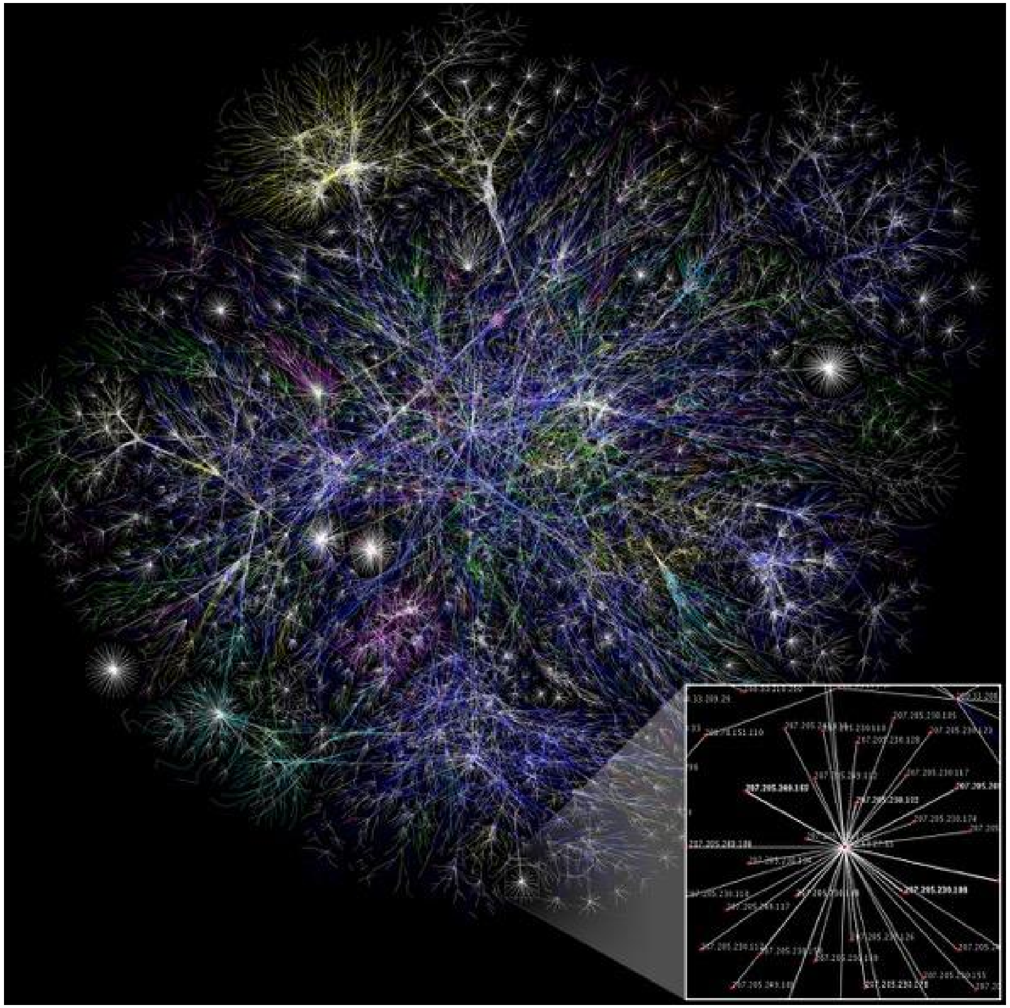
(c)

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

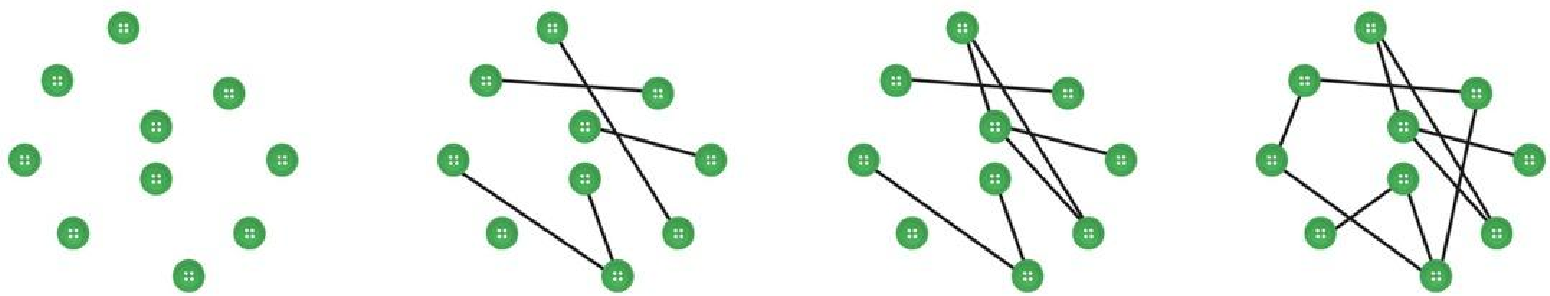
交通网络（美国航线）

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）。

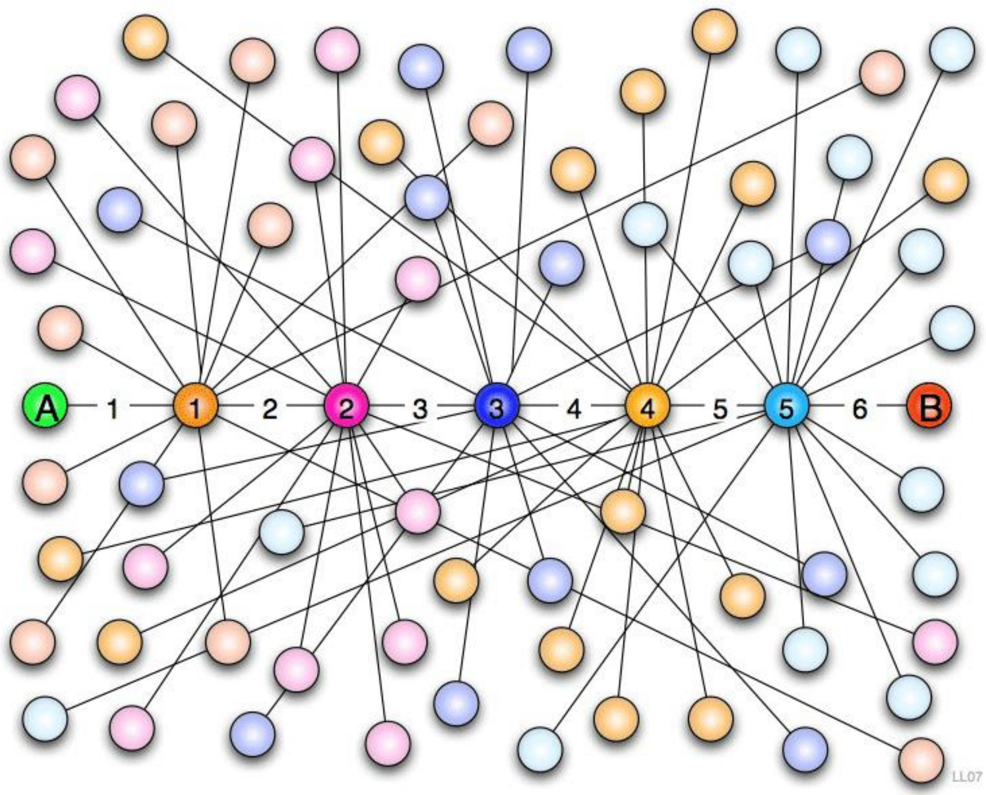
他们提出了ER随机图模型：在给定N个顶点后，规定每两个顶点之间都有p的概率连起来（$0 \le p \le 1$）,而且这些判定之间两两无关。



$p = 0$ $p = 0.1$ $p = 0.15$ $p = 0.25$

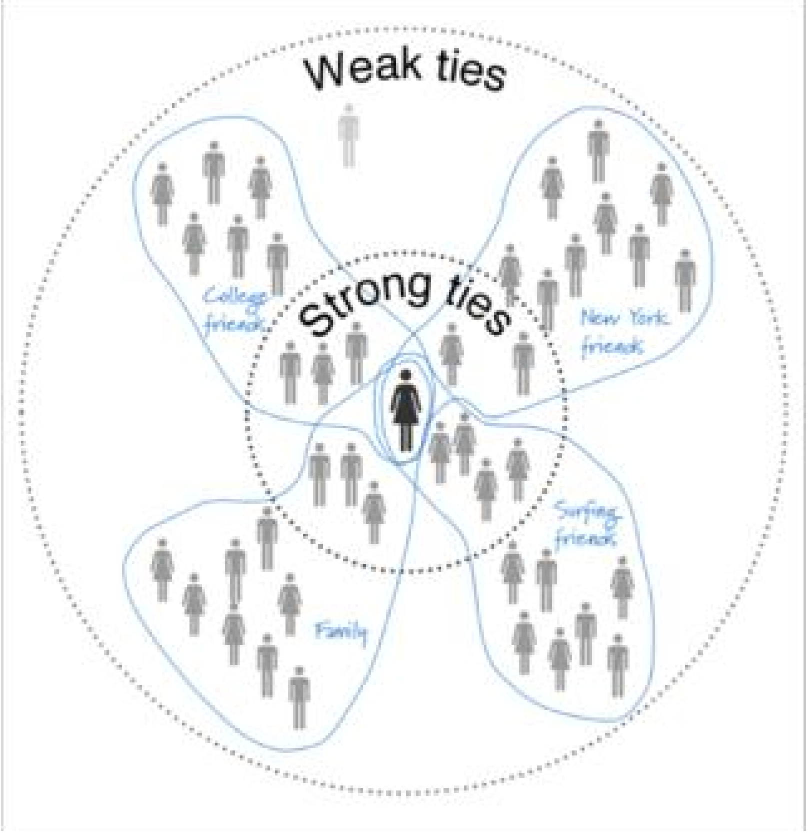
图的应用：小世界理论

六度分离哈佛大学著名的社会心理学家Stanley Mailgrams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

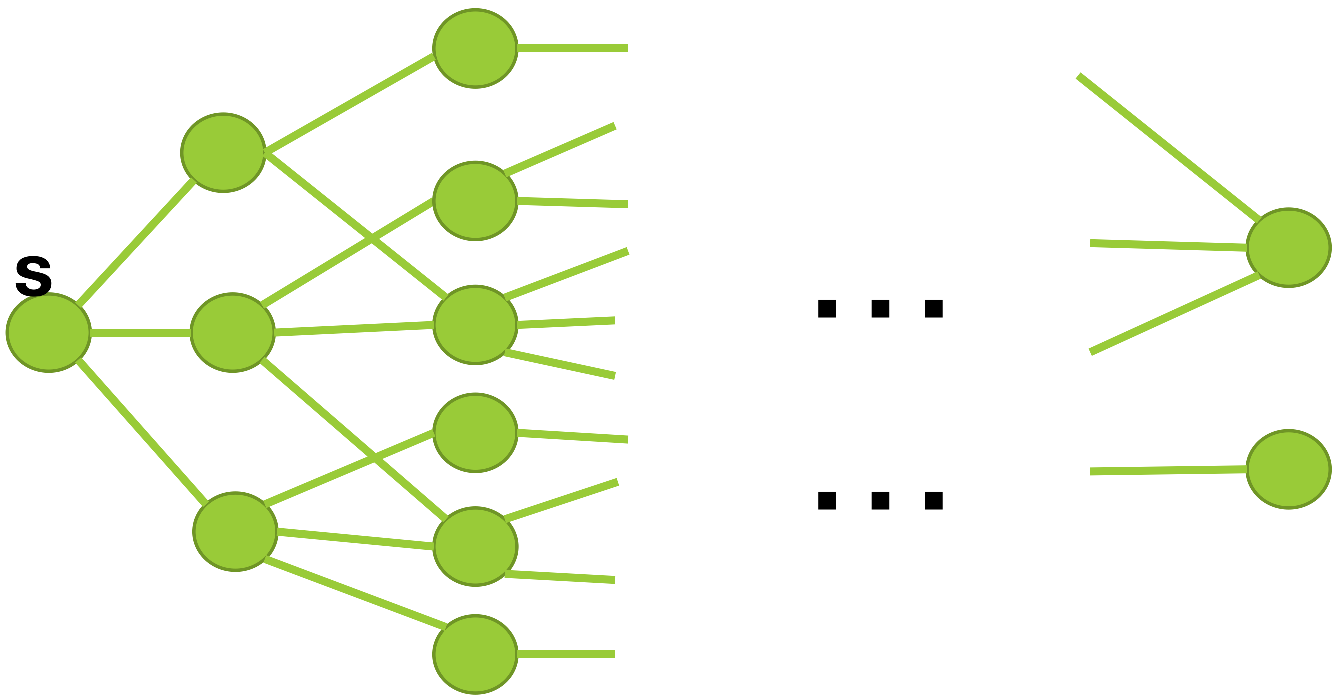
2 广度优先搜索

3 深度优先搜索

4 深度优先搜索应用

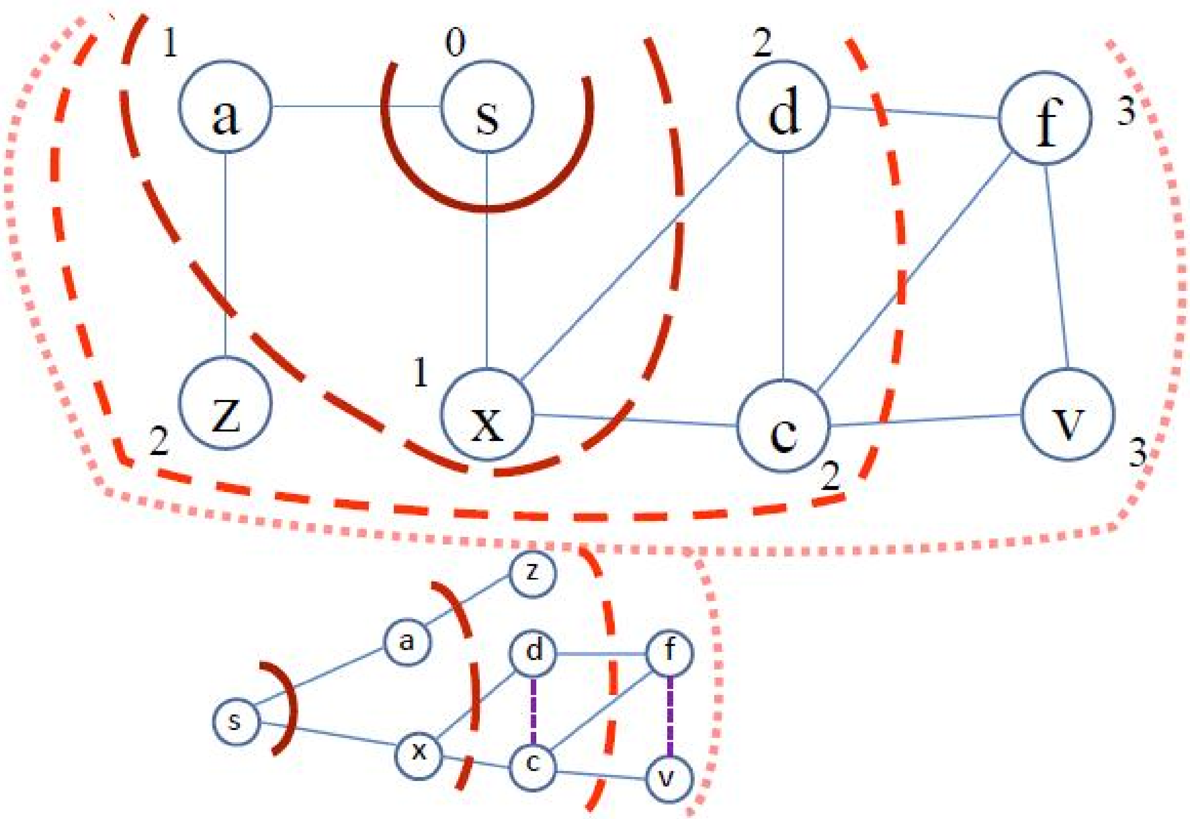
2 广（宽）度优先搜索

图搜索（Graph Search），求图上从源点出发到其他点的路径。



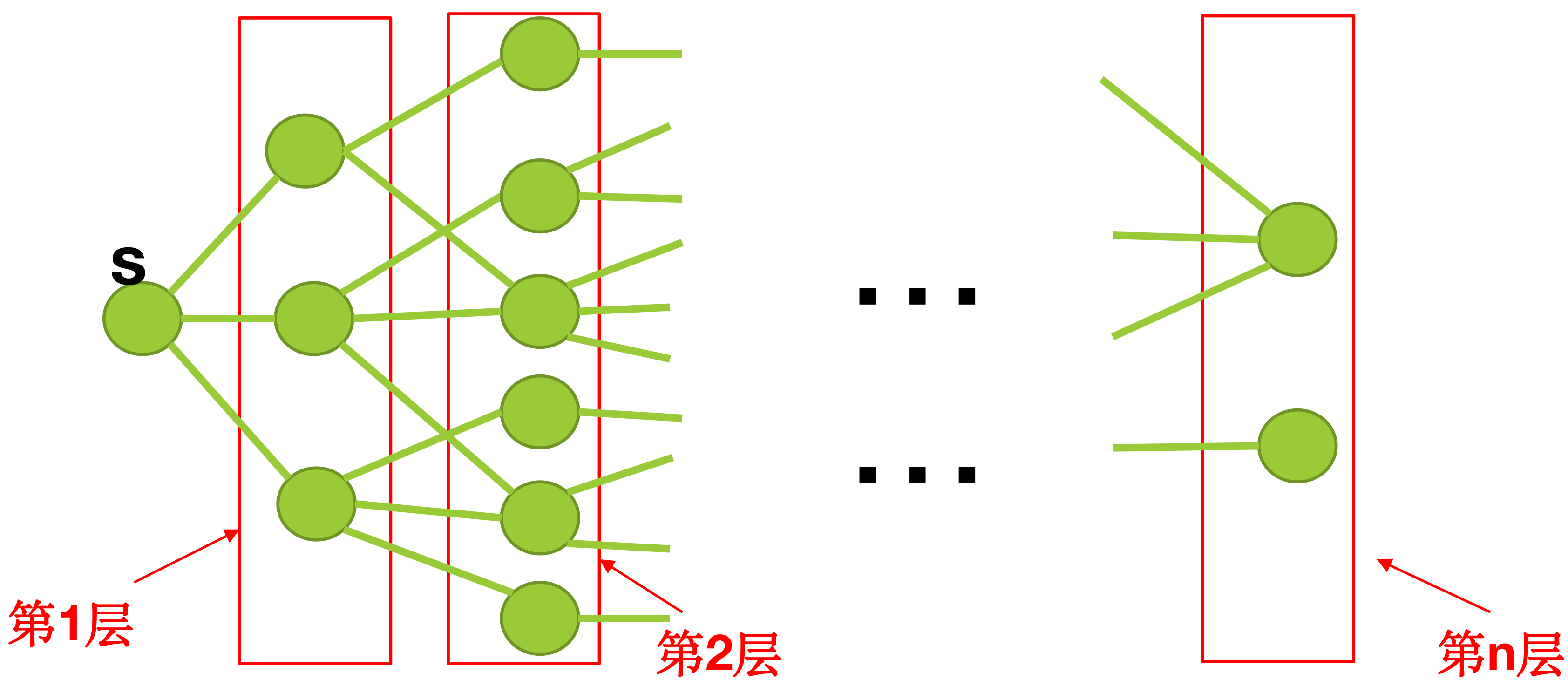
2 广（宽）度优先搜索

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。



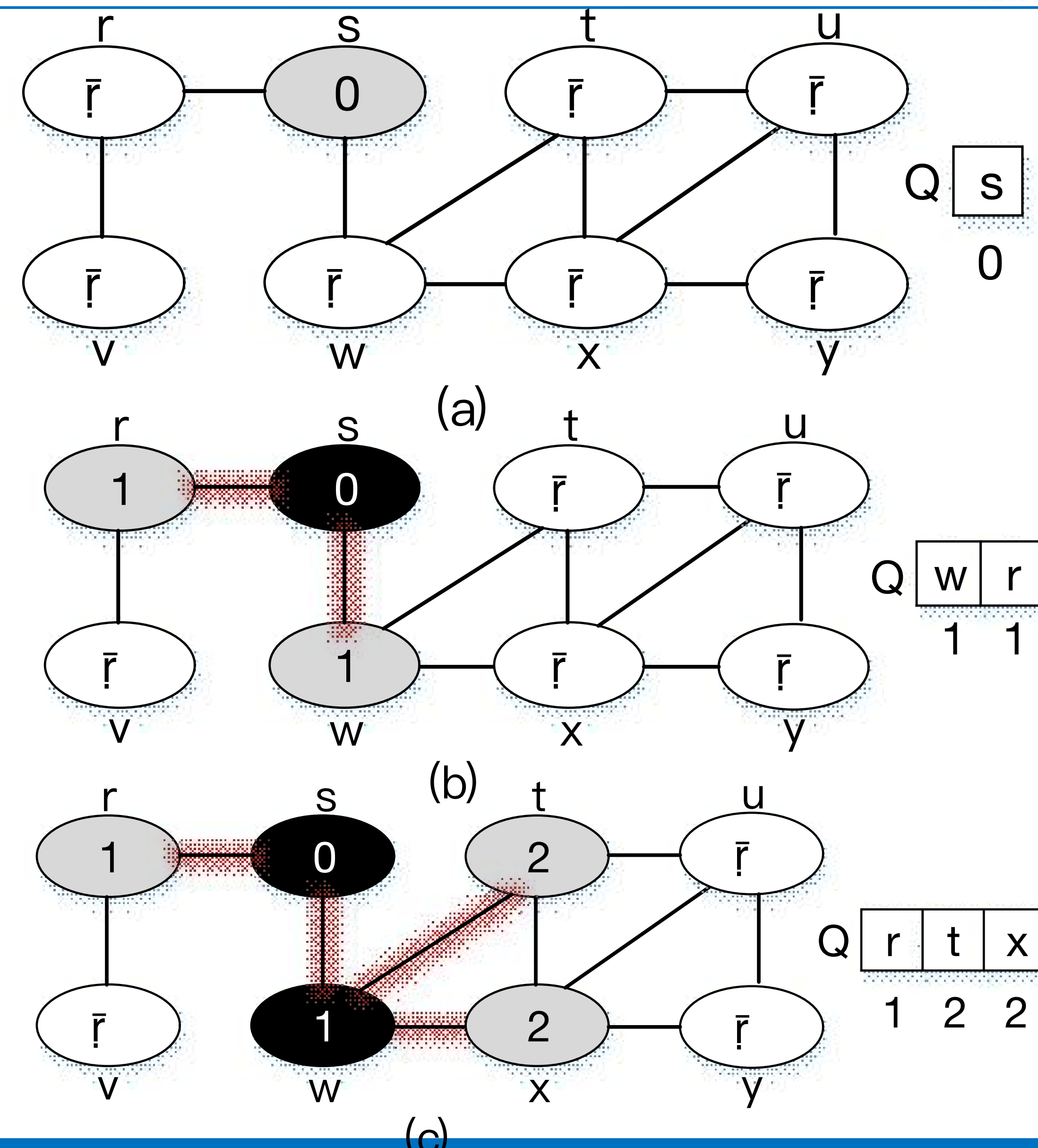
2 广（宽）度优先搜索

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)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

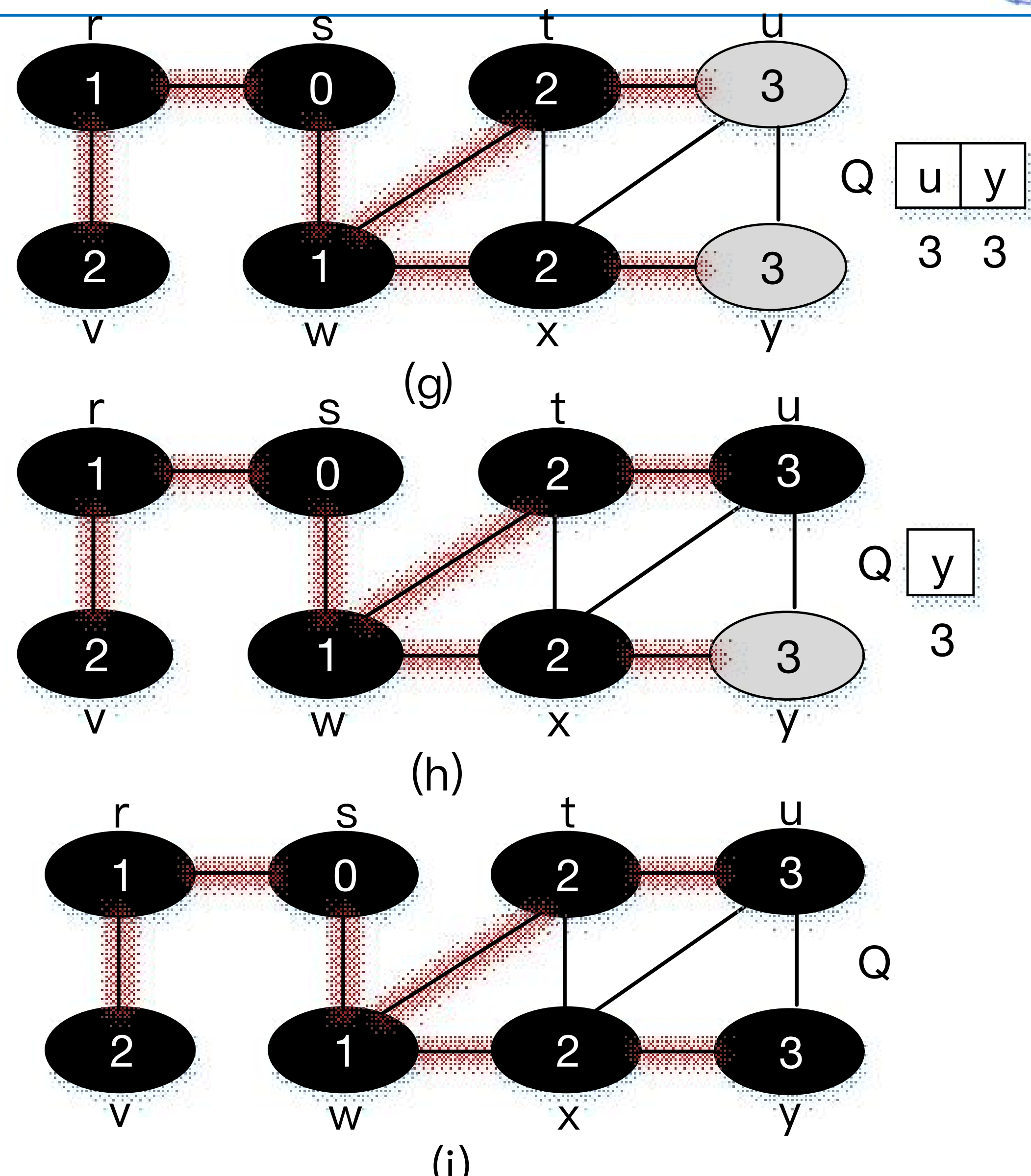


2 广度优先搜索-BFS实现过程

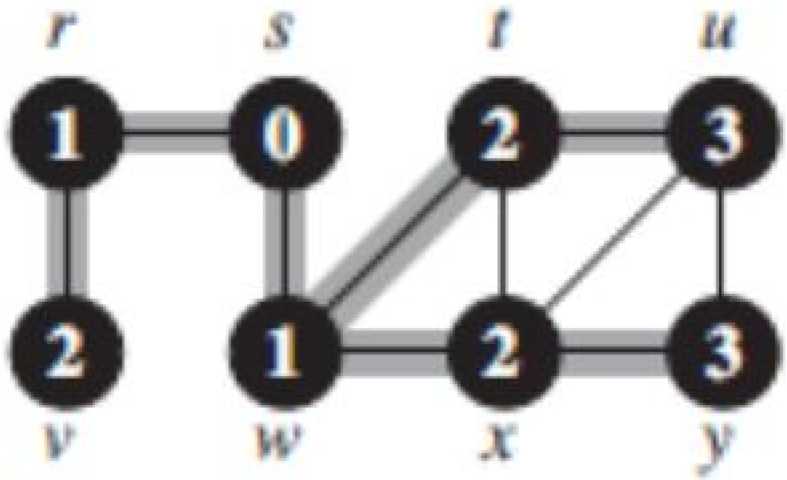
```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

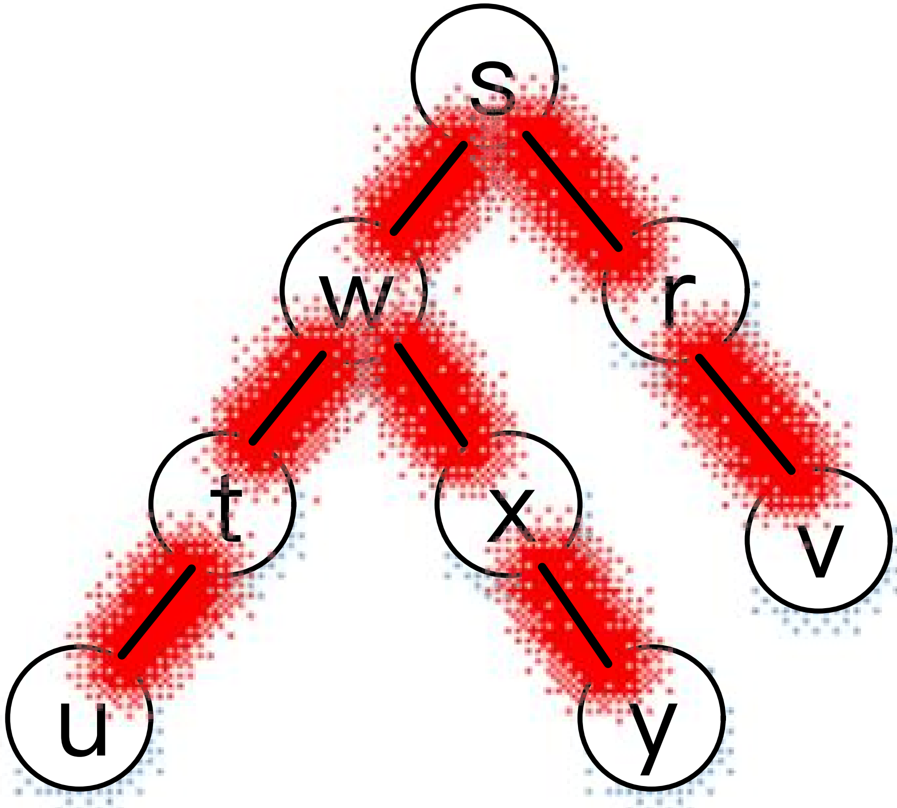
2 广度优先搜索-BFS实现过程

```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = 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)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = 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δ(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∈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∈V为G的任意一个结点，则对任意边(u,v)∈E，$\delta(s,v) \le \delta(s,u)+1$

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δ(s,v)

下面我们首先证明v.d是δ(s,v)的上界

引理2

设G=(V,E)是一个有向或无向图，并假设算法BFS从G中一已知源结点s∈V开始执行，在执行终止时，对每个顶点v∈V，变量v.d的值满足：v.d≥δ(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<=n时，假设所有点v都成立；当L=n+1时，假设s到v路径为$v_0e_1v_1e_2$ ... $e_nv_ne_{n+1}v_{n+1}$，根据归纳假设v_n加入queue的时候v_n.d等于或小于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< =n+1

2广度优先搜索-最短路径δ(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$<v_1,v_2,\ldots,v_r>$，其中v_1是队列Q的头，v_r是队列的尾。那么$v_r.d \le v_1.d+1$且$v_i.d \le v_{i+1}.d$, i=1,2,..,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$v_r.d \le v_1.d+1 \le v_2.d+1$，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δ(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∈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∈V，在运行终止时，对任意v∈V，v.d=δ(s,v)。此外，对任意从s可达的节点v≠s，从s到v的最短路径之一是从s到v. π的最短路径再加上边(v. π,v)。

2广度优先搜索-最短路径δ(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证明：

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

根据引理2，v.d≥δ(s,v)，于是v.d大于δ(s,v)，注意v必然可达，否则δ(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δ(s,v)=δ(s,u)+1。同时，δ(s,v)最小，所以δ(s,u)=u.d。于是，我们有v.d>δ(s,v)=δ(s,u)+1=u.d+1。

这个不可能的

提 纲

1 图的基本概念

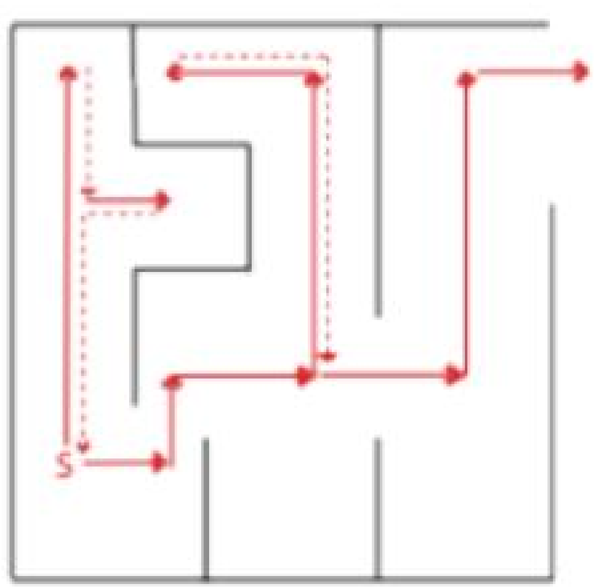
2 广度优先搜索

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

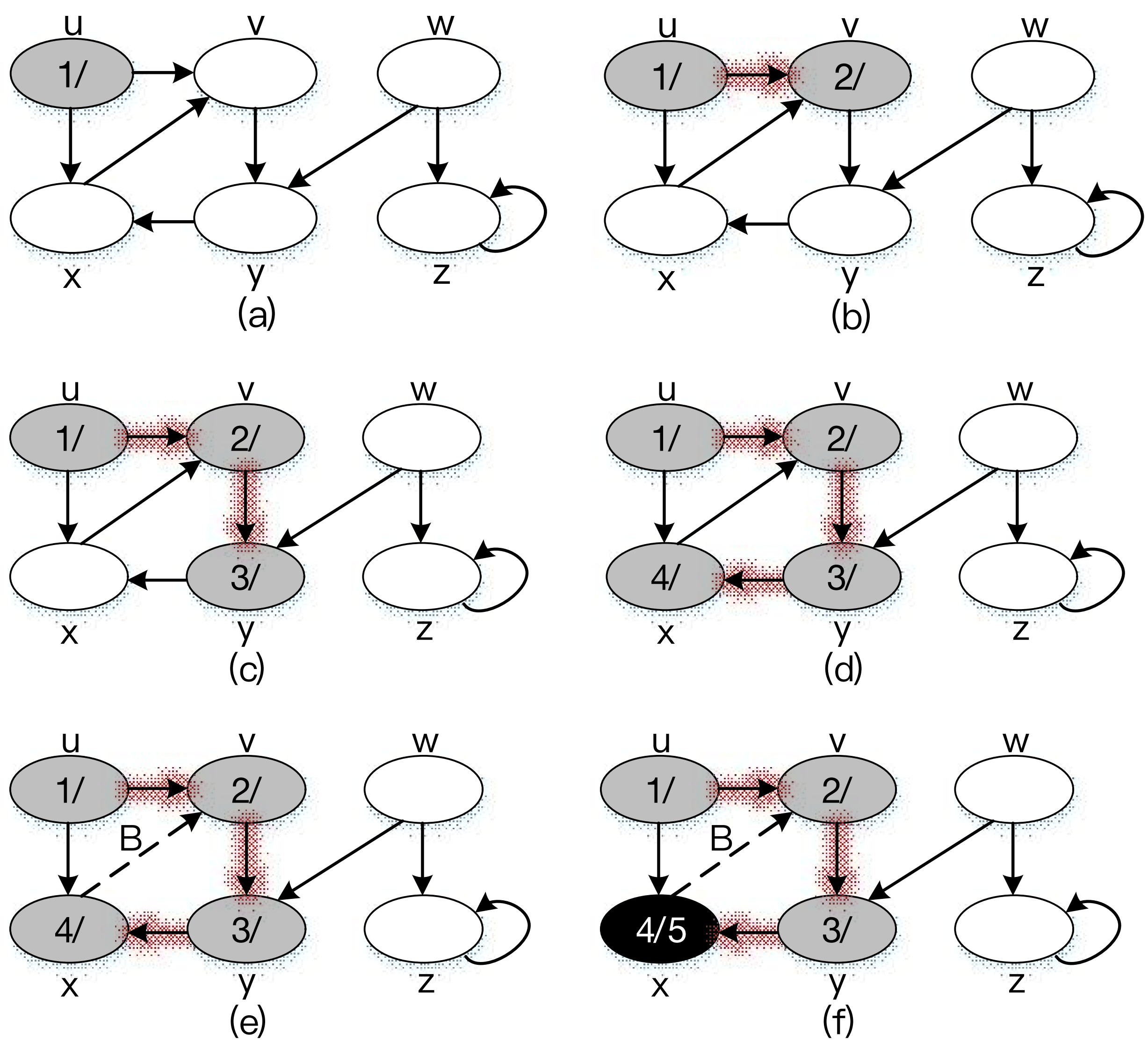
- 令$G=(V, E)$是一个有向图或无向图，搜索时沿着树的深度遍历树的结点，尽可能深的搜索树的分支。
- 当结点u的所有边都已被搜索过，搜索将回溯到发现结点u的那条边的起始结点。这一过程一直进行到已发现从源结点可达的所有结点为止。
- 如果还存在未被发现的结点，则选择其中一个作为源结点并重复以上过程，整个进程反复进行直到所有结点都被访问为止。

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

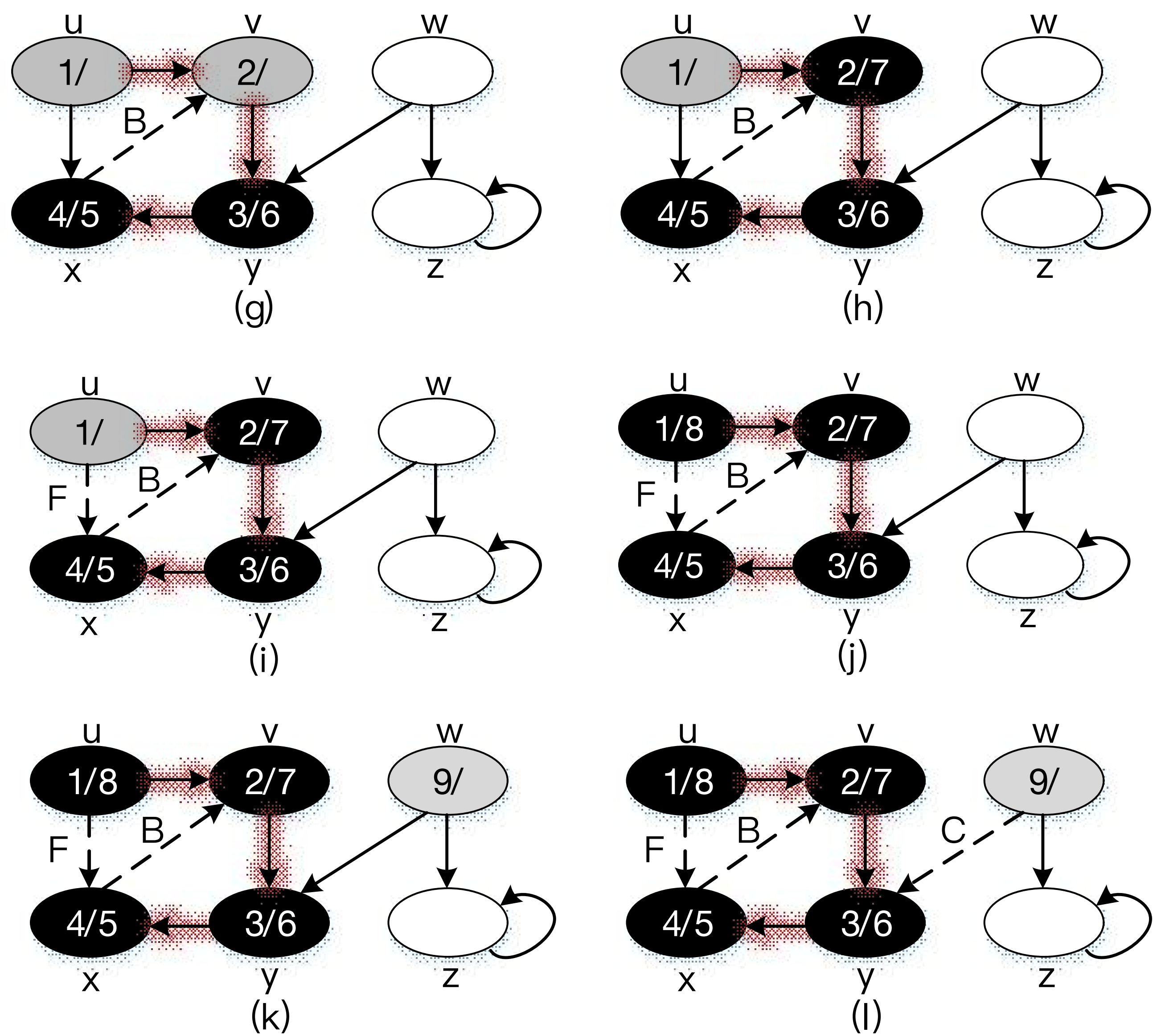
```
DFS(G)
for each vertex u ∈ G.V
        u.color = WHITE
        u.π = NIL
time = 0
for each vertex u ∈ G.V
        if u.color == WHITE
                DFS-VISIT(G, u)

DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
        if v.color == WHITE
                v.π = u
                DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```



3深度优先搜索-DFS实现过程：递归

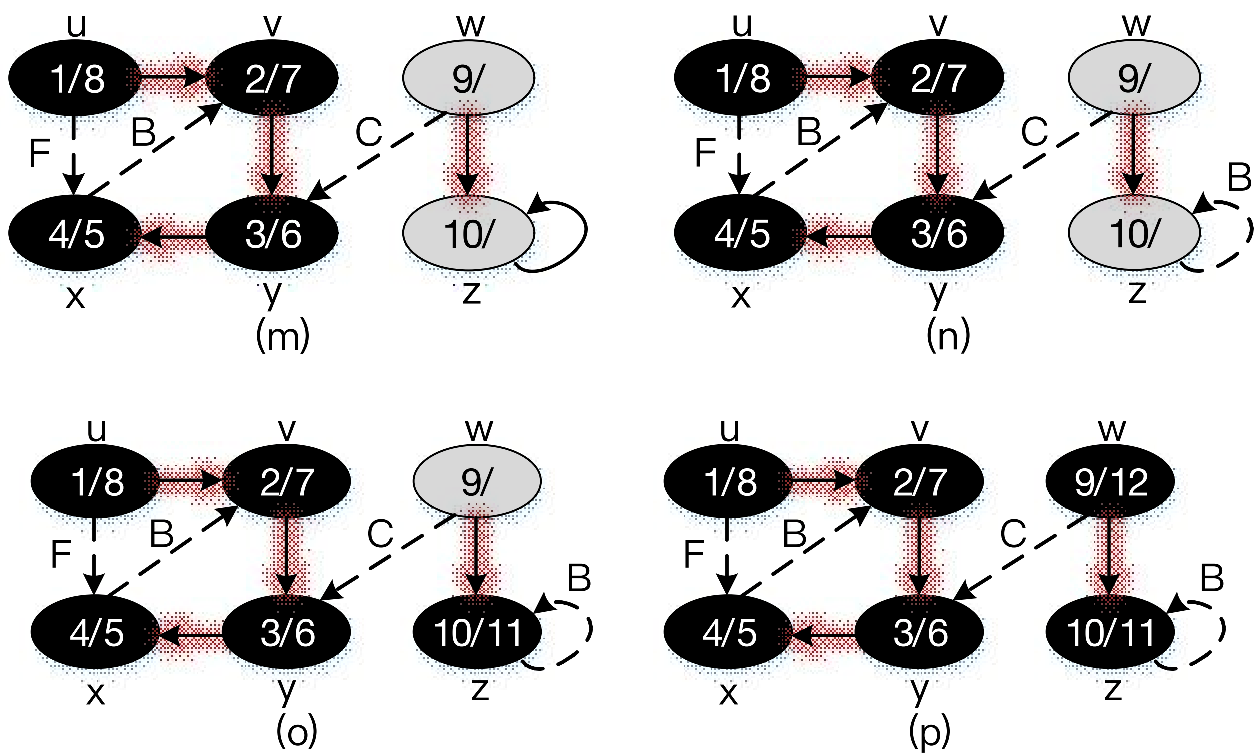
```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```



3深度优先搜索-DFS实现过程：递归

```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)

DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```



3深度优先搜索-DFS时间复杂度分析

```
DFS(G)
1  for each vertex u ∈ G.V
2      u.color = WHITE
3      u.π = NIL
4  time = 0
5  for each vertex u ∈ G.V
6      if u.color == WHITE
7          DFS-VISIT(G, u)

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

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中第4-7行语句占用的整个时间应为θ(E)

DFS的运行时间为θ(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

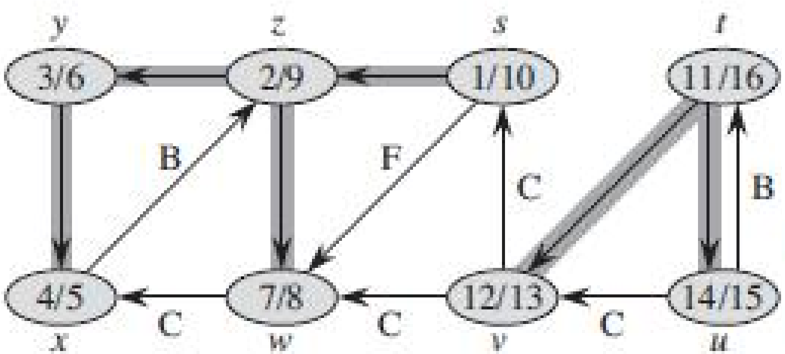
不同点

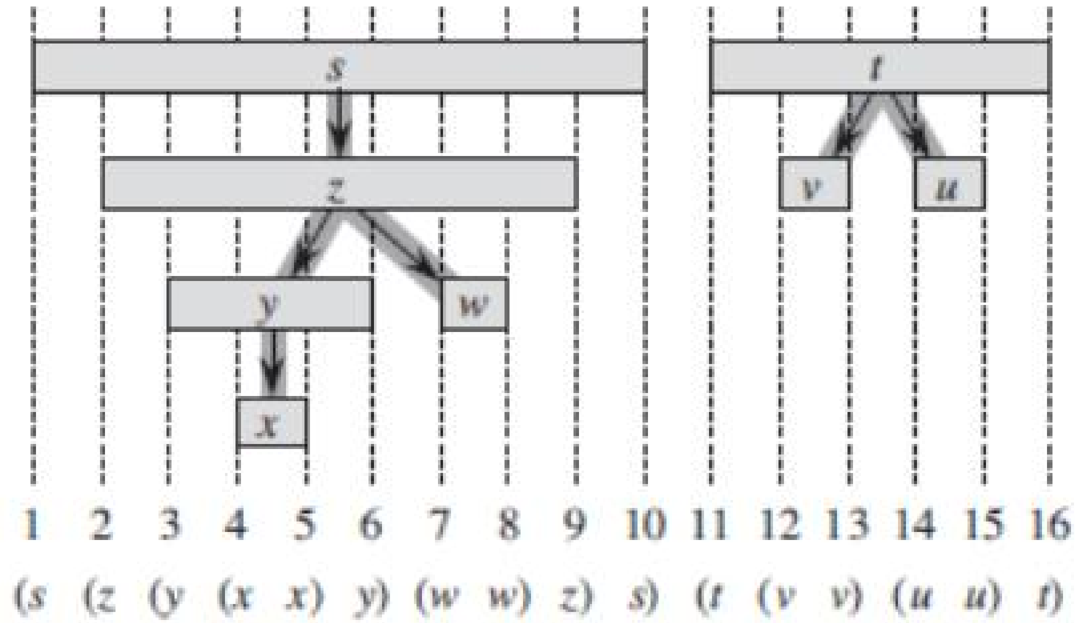
BFS使用队列；DFS使用栈

BFS按层遍历图；DFS贪心地搜索

3深度优先搜索-性质

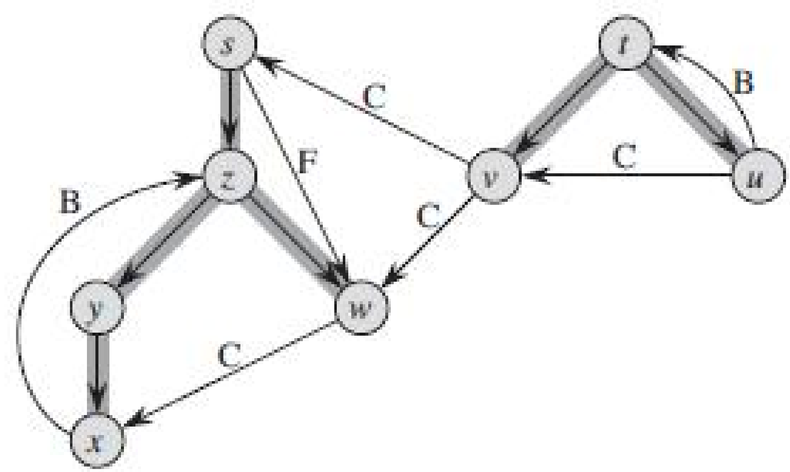
从不同的点出发或者沿着不同点深优，会得到不同深度优先搜索森林。





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颗树，弱连通则多颗树。



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，结节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≤u.f，因为v∈Adj[w]，对DFS_Visit(w)的调用保证完成w之前先完成v，因此v.f<w.f≤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

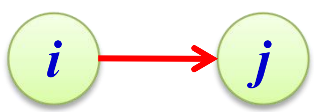
3 深度优先搜索

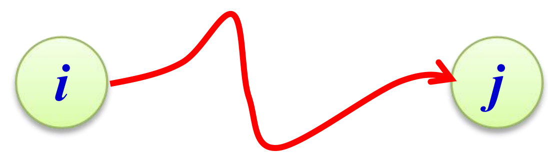
4 深度优先搜索应用

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，…，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<*i*，*j*>是图中的边（或从顶点i ⇨ j有一条路径）：



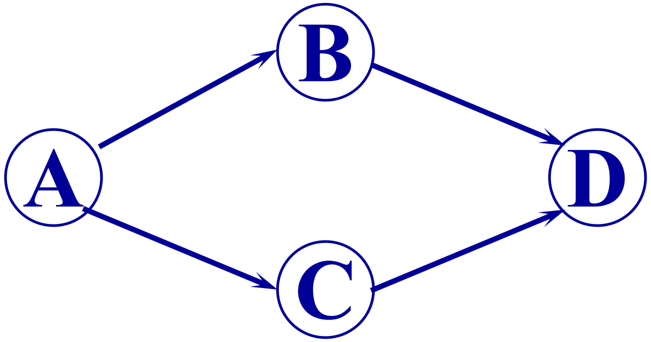


则在拓扑序列中顶点i必须排在顶点j之前。

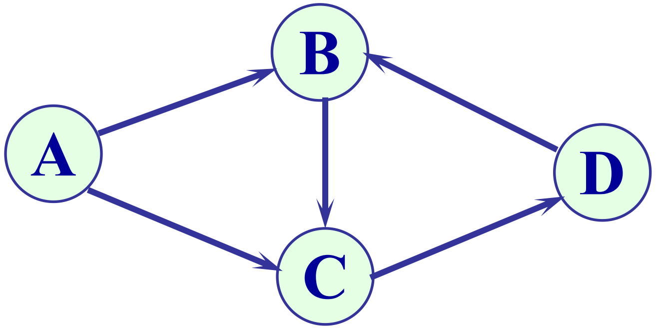
在一个有向图中找一个拓扑序列的过程称为拓扑排序。

4.1 拓扑排序

有向无环图(DAG：Directed Acyclic Graph)



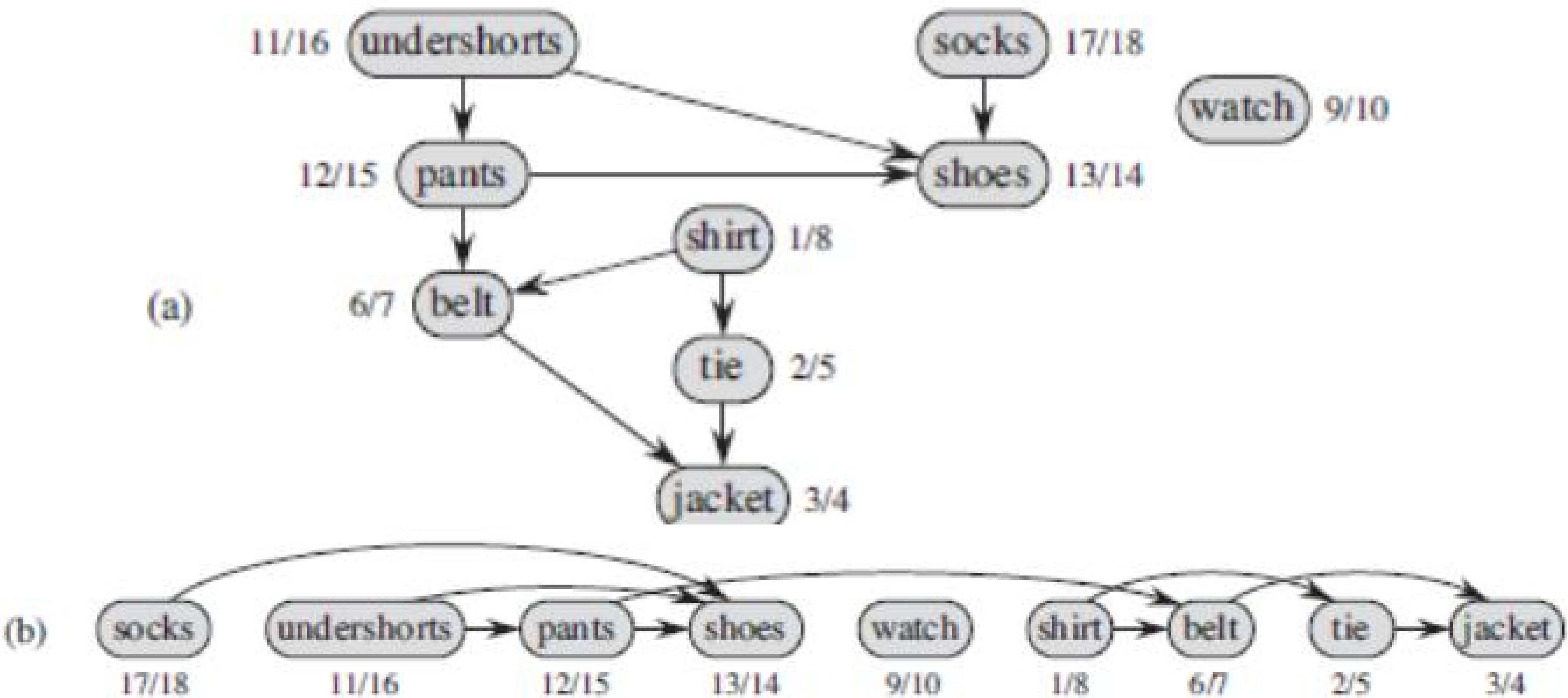
可求得拓扑有序序列：A B C D　或　A C B D



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。



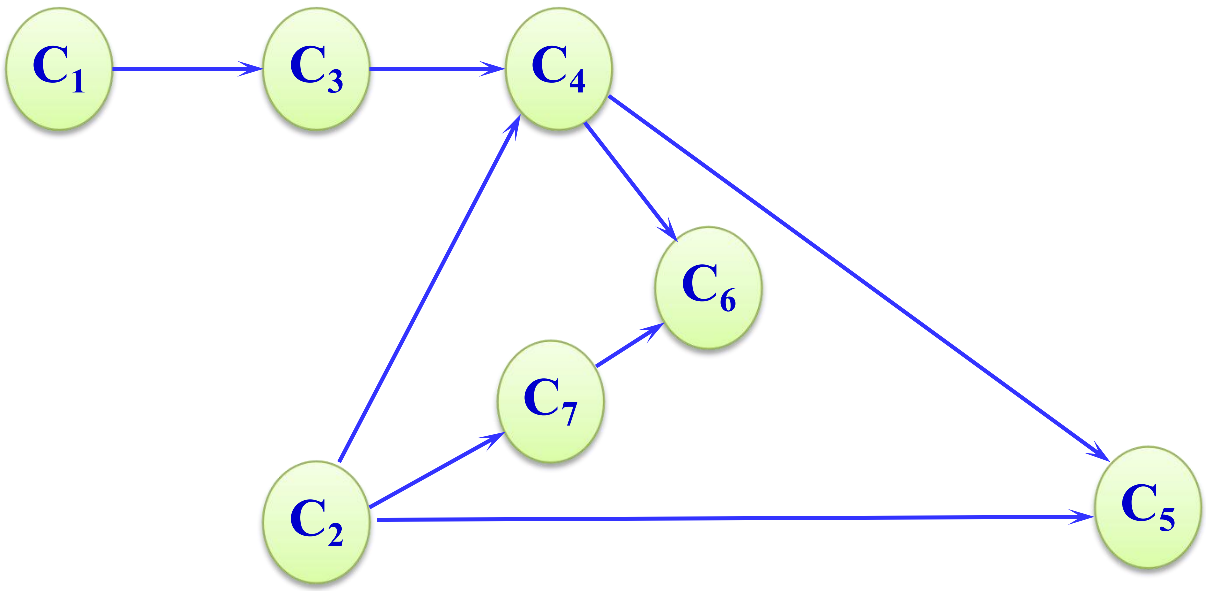
4.1 拓扑排序

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_1	高等数学	无
C_2	程序设计	无
C_3	离散数学	C_1
C_4	数据结构	C_2，C_3
C_5	编译原理	C_2，C_4
C_6	操作系统	C_4，C_7
C_7	计算机组成原理	C_2

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

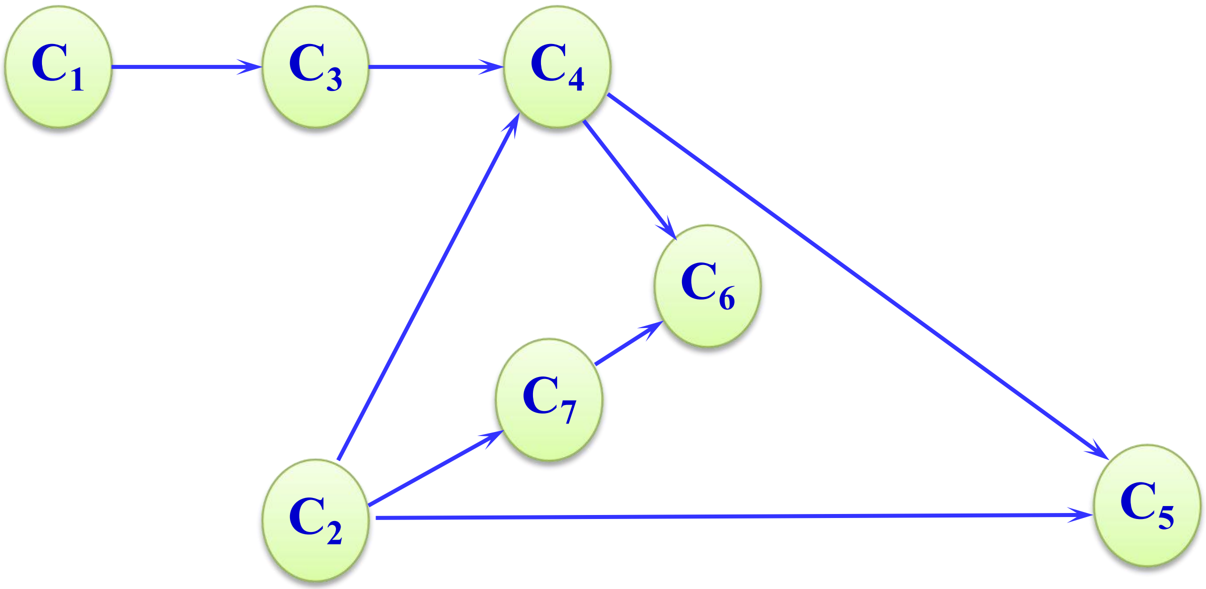
（1）从有向图中选择一个没有前驱（即入度为0）的顶点并且输出它。

（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

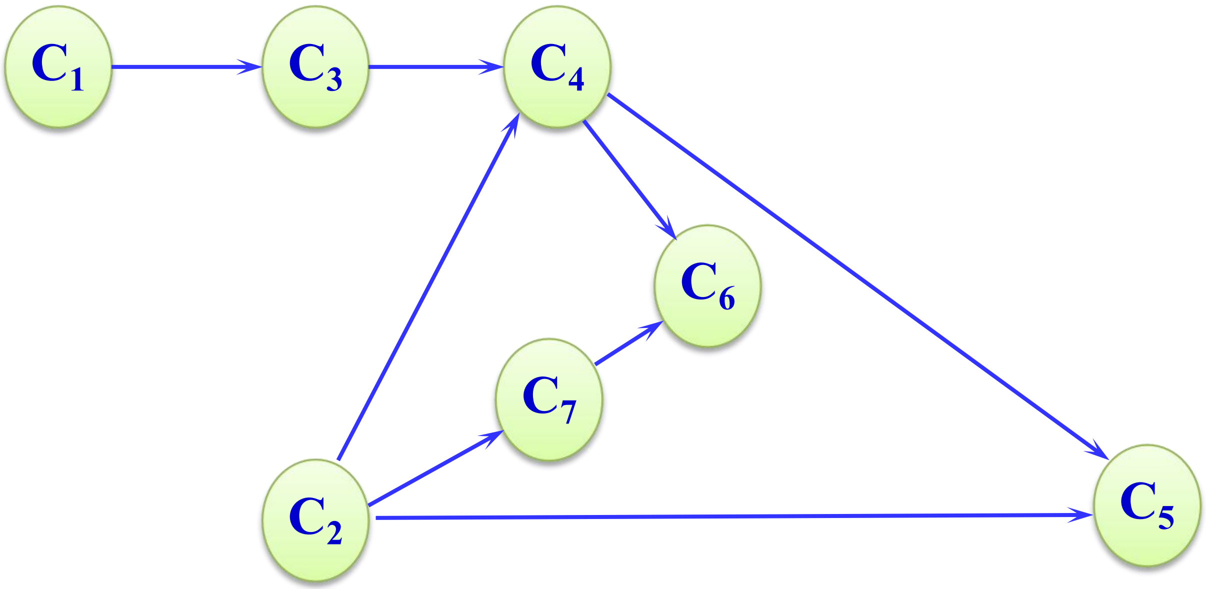


产生一个拓扑序列：

C_1 C_2 C_3 C_4 C_5 C_7 C_6

排序完成

拓扑排序方法1演示



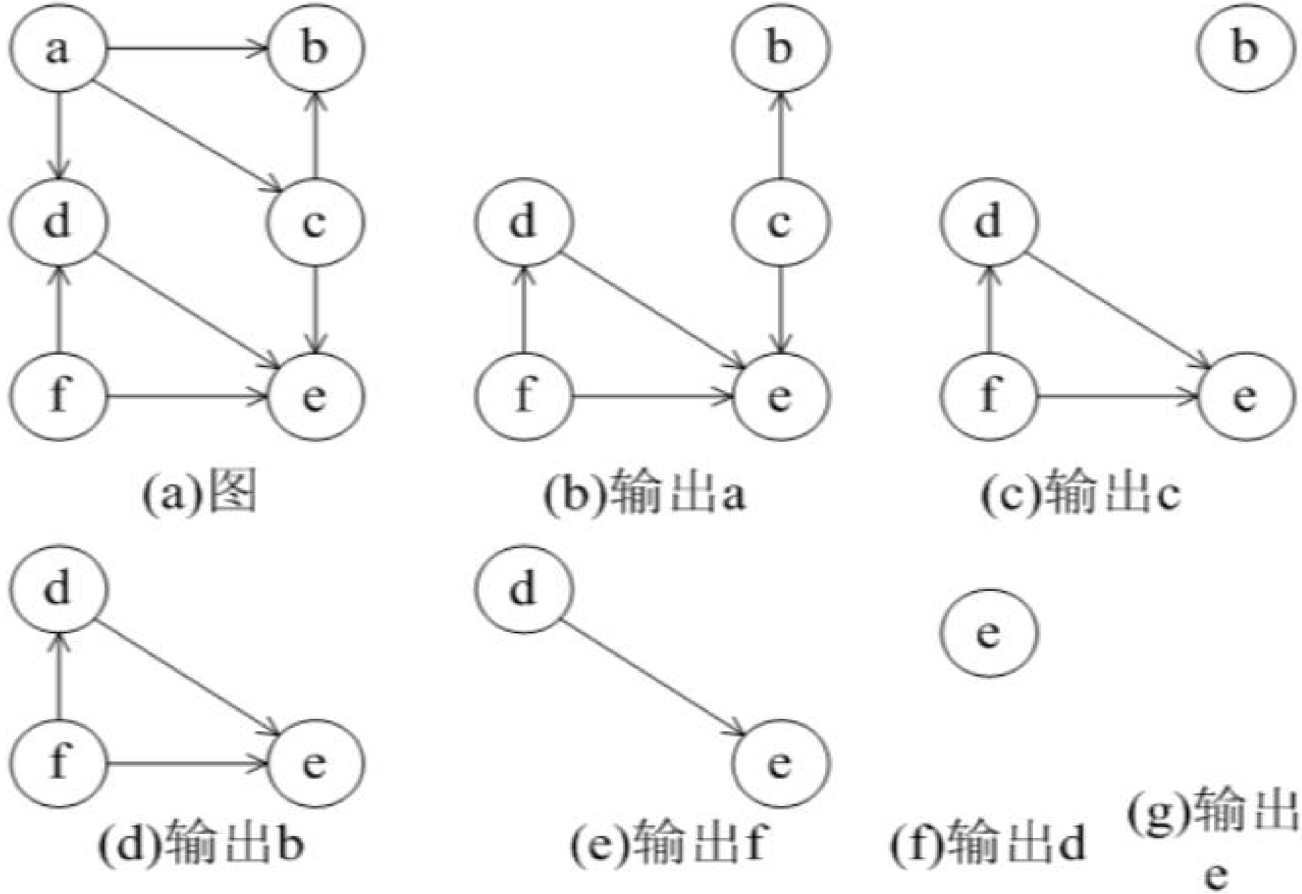
产生一个拓扑序列：

C_1 C_2 C_3 C_4 C_5 C_7 C_6

排序完成

4.1 拓扑排序

拓扑排序方法1 演示



拓扑排序序列：acbfde

4.1 拓扑排序

拓扑排序方法2：基于DFS

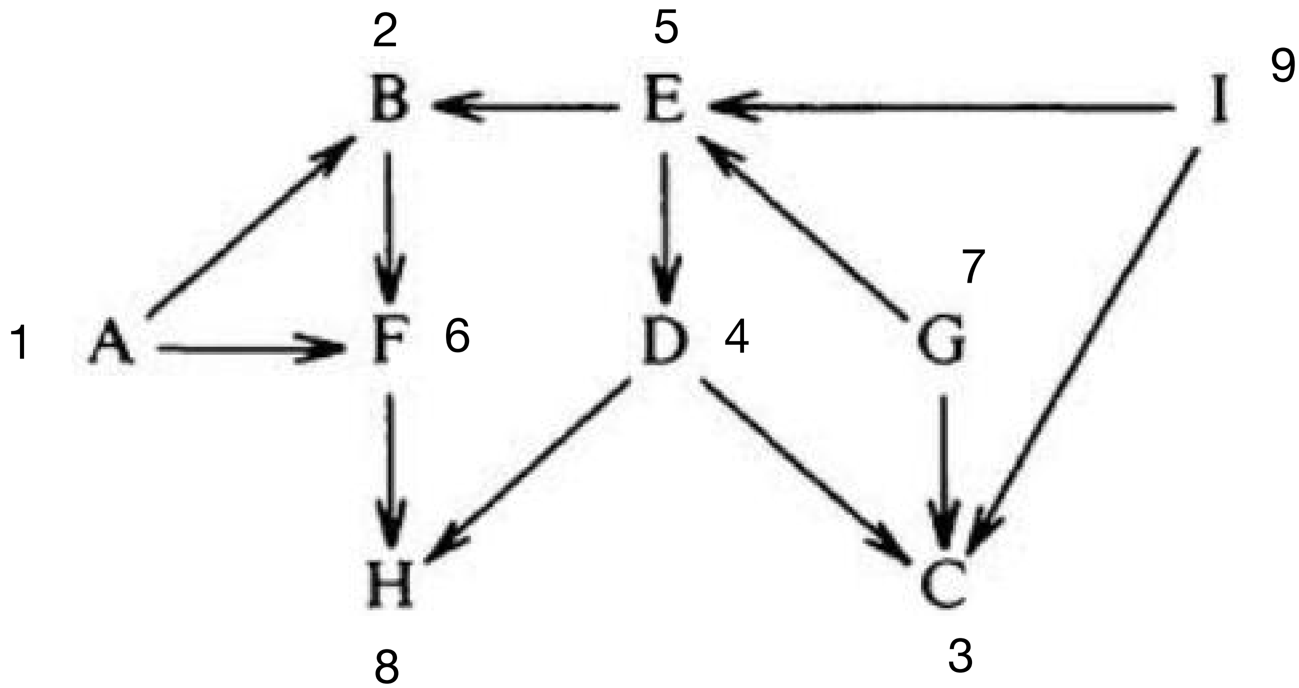
```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

```
TOPOLOGICAL-SORT(G)
call DFS(G) to compute finishing times v.f for each vertex v
as each vertex is finished, insert it onto the front of a linked list
return the linked list of vertices
```

4.1 拓扑排序

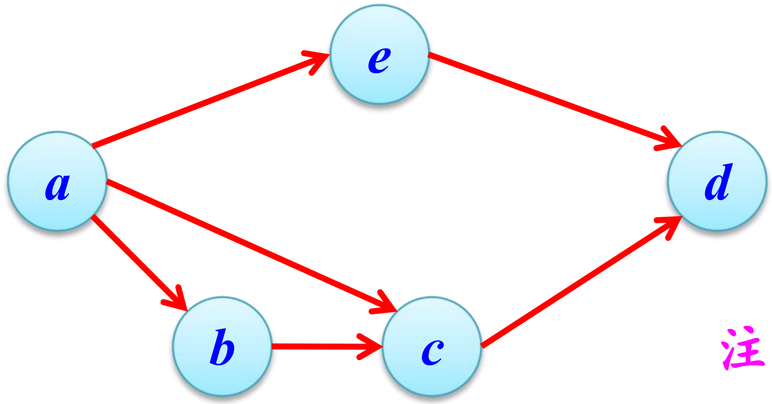
拓扑排序方法2：基于DFS



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

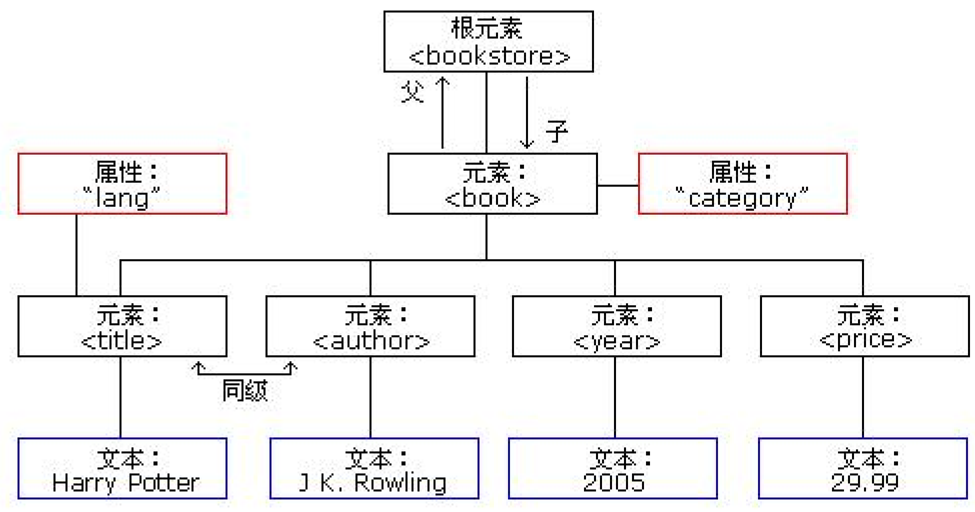
XML 文档形成了一种树结构，它从“根部”开始，然后扩展到“枝叶”

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

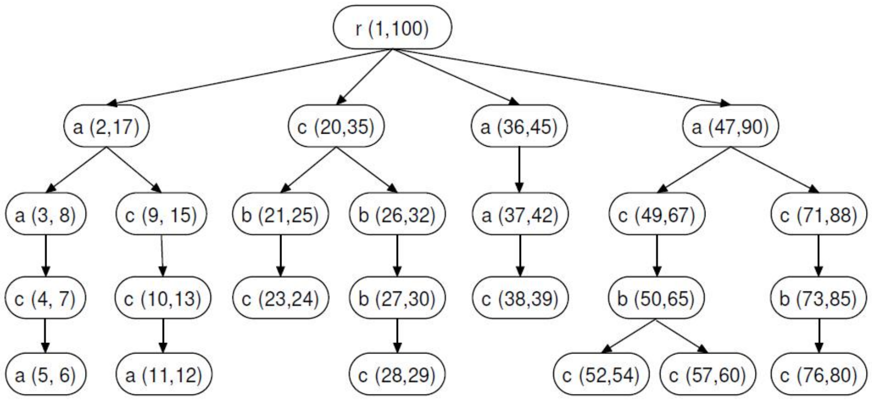
XPath 是一门在 XML 文档中查找信息的语言。XPath 用于在 XML 文档中通过元素和属性进行导航。

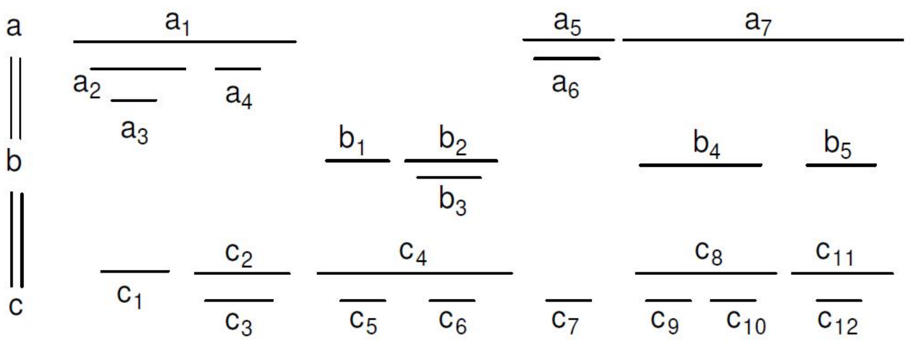
Xpath查询：//a//b//c

- Xpath查询的路径表示：

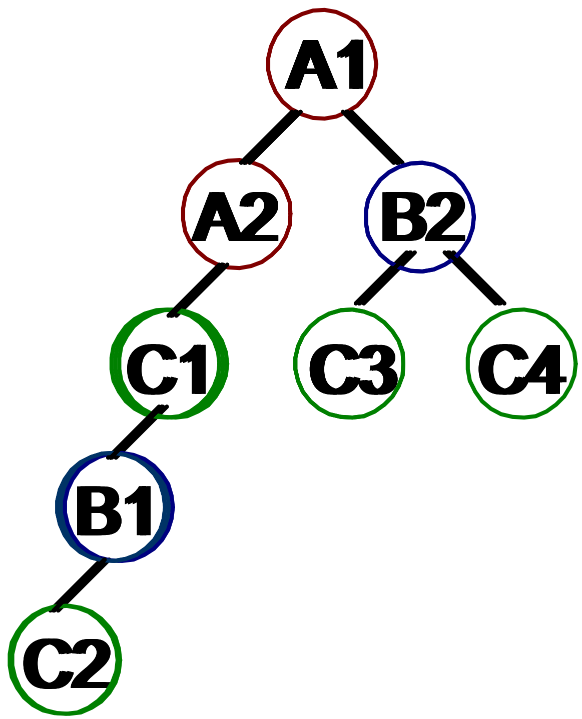
DFS编码

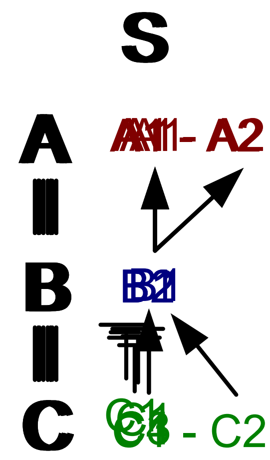
每个查询点的候选流（按照**发现时间**排序）、针对每个查询变量的栈





PathStack示例





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

时间复杂度 O(|input|+|output|)