6.图

（d）深度优先搜索

Keep it simple, stupid.

- K. Johnson

邓俊辉

deng@tsinghua.edu.cn

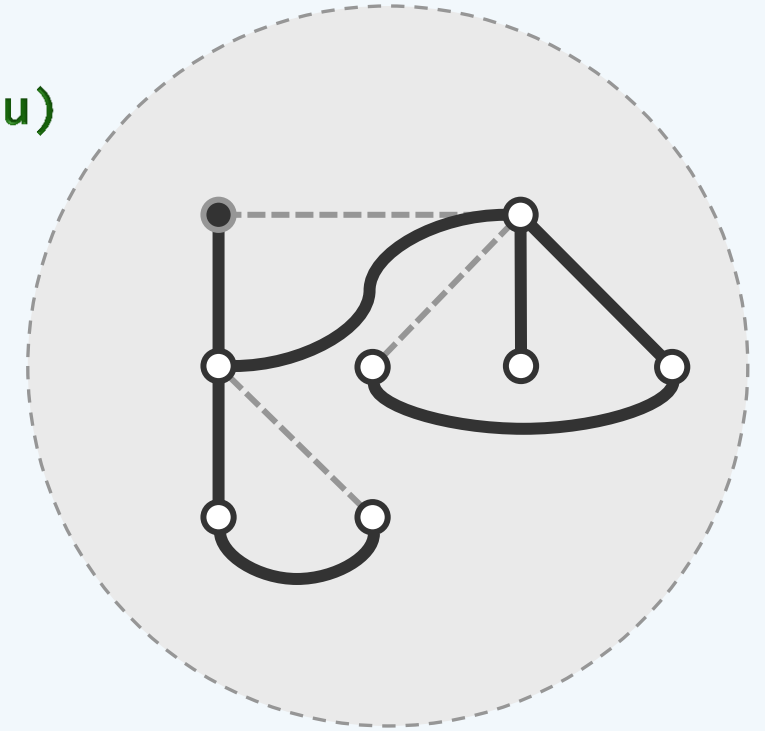
算法

❖ DFS(s) //始自顶点s的深度优先搜索（Depth-First Search）

访问顶点s

若s尚有未被访问的邻居，则任取其一u，递归执行DFS(u)

否则，返回



❖ 若此时图中尚有顶点未被访问 //何时出现这一情况？

任取这样的一个顶点作起始点

重复上述过程

直至所有顶点都被访问到

❖ 等效于树的先序遍历

事实上，DFS也的确会构造出原图的一棵支撑树（DFS tree）

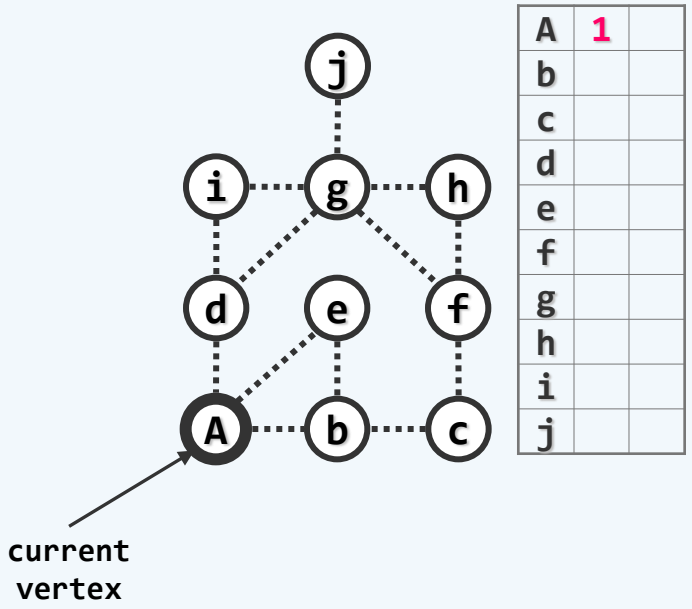
Graph::DFS()

```
template <typename Tv, typename Te> //顶点类型、边类型
void Graph<Tv, Te>::DFS( int v, int & clock ) {
    dTime(v) = ++clock; status(v) = DISCOVERED; //发现当前顶点v
    for ( int u = firstNbr(v); -1 < u; u = nextNbr(v, u) ) //枚举v的每一邻居u
        /* ... 视u的状态，分别处理 ... */
        /* ... 与BFS不同，含有递归 ... */
    status(v) = VISITED; fTime(v) = ++clock; //至此，当前顶点v方告访问完毕
}
```

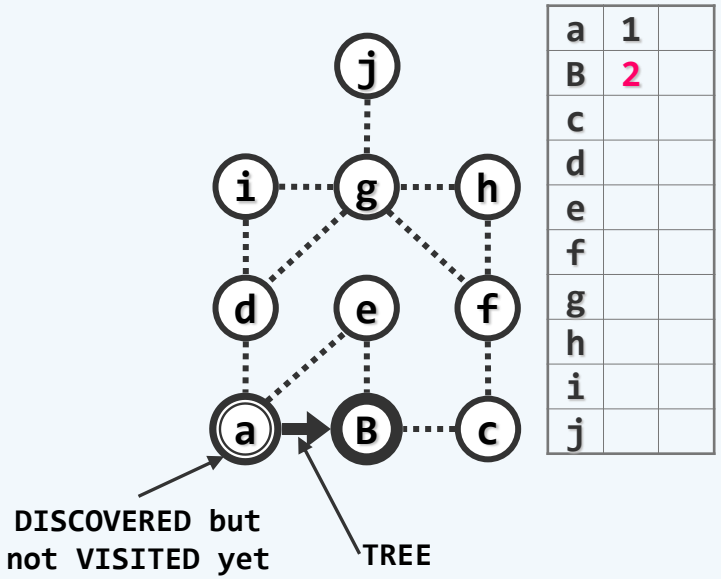
Graph::DFS()

```
❖ for ( int u = firstNbr(v); -1 < u; u = nextNbr(v, u) ) //枚举v所有邻居u
    switch ( status(u) ) { //并视其状态分别处理
      case UNDISCOVERED: //u尚未发现，意味着支撑树可在此拓展
        status(v, u) = TREE; parent(u) = v; DFS(u, clock); break; //递归
      case DISCOVERED: //u已被发现但尚未访问完毕，应属被后代指向的祖先
        status(v, u) = BACKWARD; break;
      default: //u已访问完毕（VISITED，有向图），则视承袭关系分为前向边或跨边
        status(v, u) = dTime(v) < dTime(u) ? FORWARD : CROSS; break;
    } //switch
```

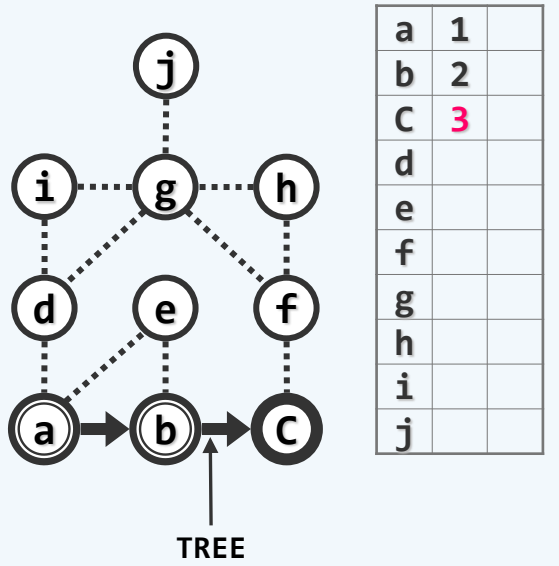
实例（无向图）



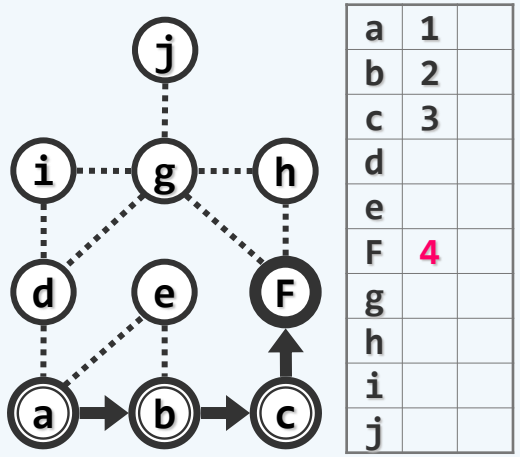
A	1	
b		
c		
d		
e		
f		
g		
h		
i		
j		



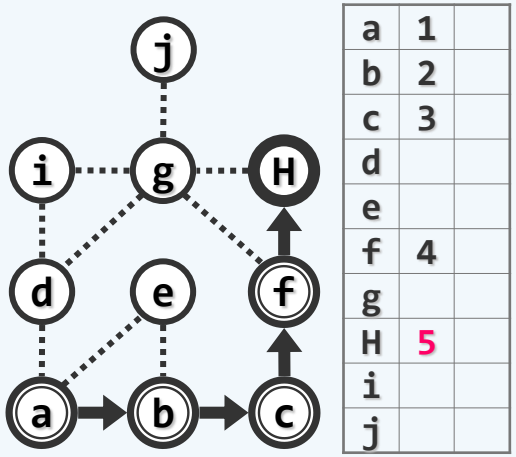
a	1	
B	2	
c		
d		
e		
f		
g		
h		
i		
j		



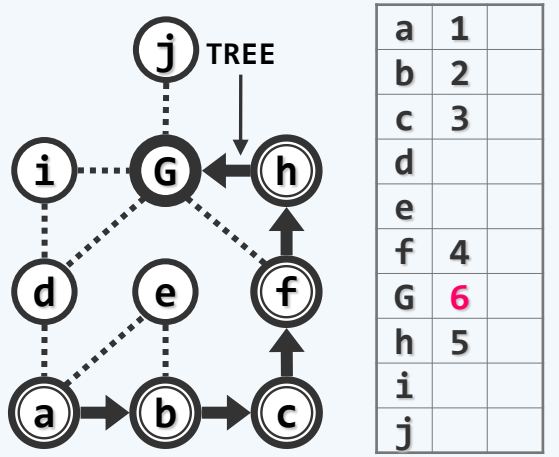
a	1	
b	2	
C	3	
d		
e		
f		
g		
h		
i		
j		



a	1	
b	2	
c	3	
d		
e		
F	4	
g		
h		
i		
j		

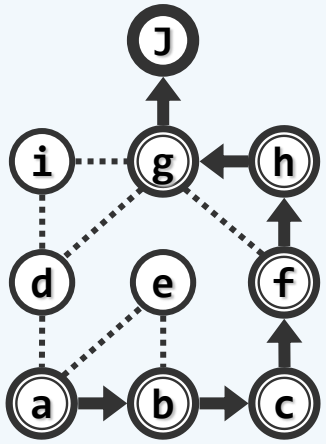


a	1	
b	2	
c	3	
d		
e		
f	4	
g		
H	5	
i		
j		

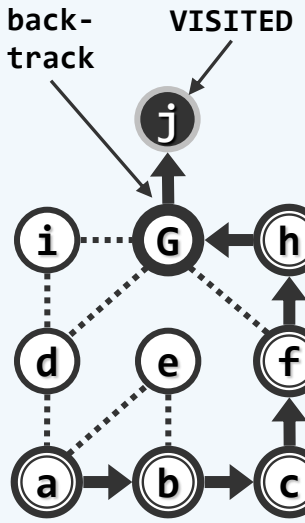


a	1	
b	2	
c	3	
d		
e		
f	4	
G	6	
h	5	
i		
j		

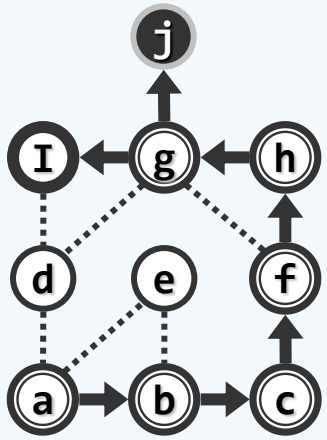
实例（无向图）



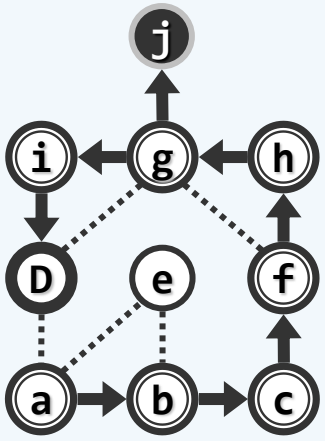
a	1	
b	2	
c	3	
d		
e		
f	4	
g	6	
h	5	
i		
J	7	



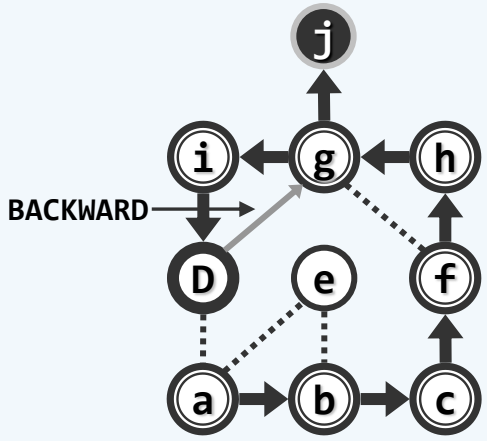
a	1	
b	2	
c	3	
d		
e		
f	4	
G	6	
h	5	
i		
j	7	8



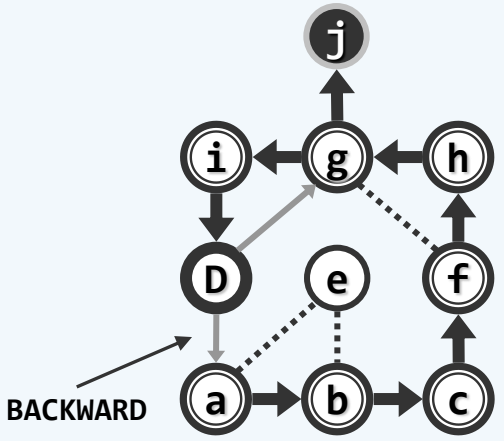
a	1	
b	2	
c	3	
d		
e		
f	4	
g	6	
h	5	
I	9	
j	7	8



a	1	
b	2	
c	3	
D	10	
e		
f	4	
g	6	
h	5	
i	9	
j	7	8

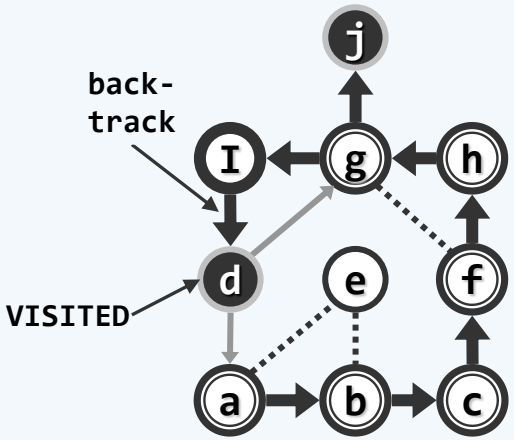


a	1	
b	2	
c	3	
D	10	
e		
f	4	
g	6	
h	5	
i	9	
j	7	8

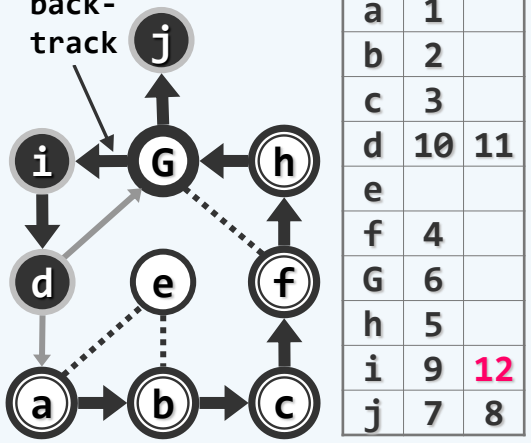


a	1	
b	2	
c	3	
D	10	
e		
f	4	
g	6	
h	5	
i	9	
j	7	8

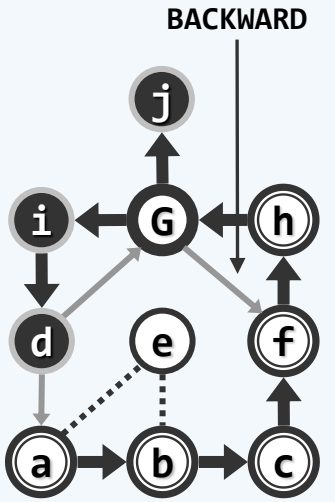
实例（无向图）



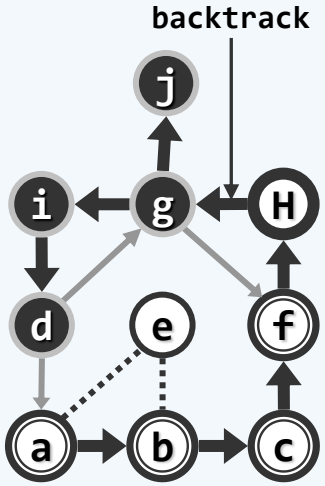
a	1	
b	2	
c	3	
d	10	11
e		
f	4	
g	6	
h	5	
I	9	
j	7	8



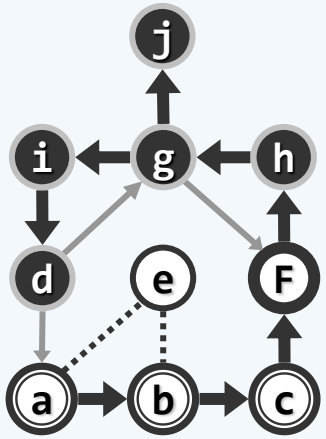
a	1	
b	2	
c	3	
d	10	11
e		
f	4	
G	6	
h	5	
i	9	12
j	7	8



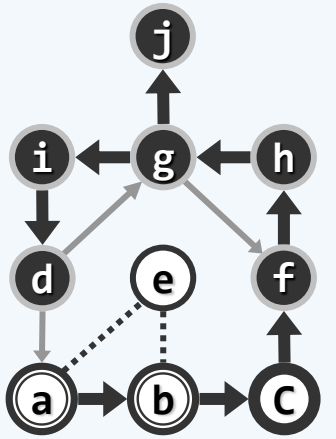
a	1	
b	2	
c	3	
d	10	11
e		
f	4	
G	6	
h	5	
i	9	12
j	7	8



a	1	
b	2	
c	3	
d	10	11
e		
f	4	
g	6	13
H	5	
i	9	12
j	7	8

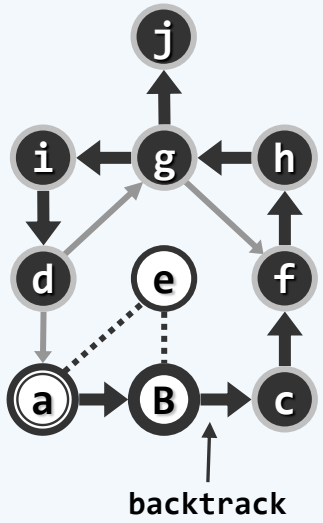


a	1	
b	2	
c	3	
d	10	11
e		
F	4	
g	6	13
h	5	14
i	9	12
j	7	8

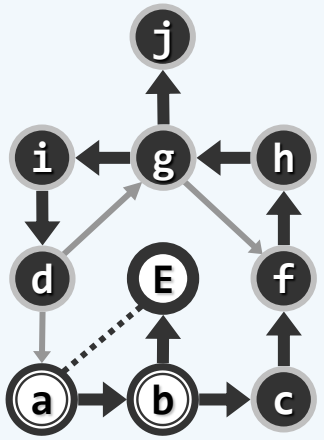


a	1	
b	2	
C	3	
d	10	11
e		
f	4	15
g	6	13
h	5	14
i	9	12
j	7	8

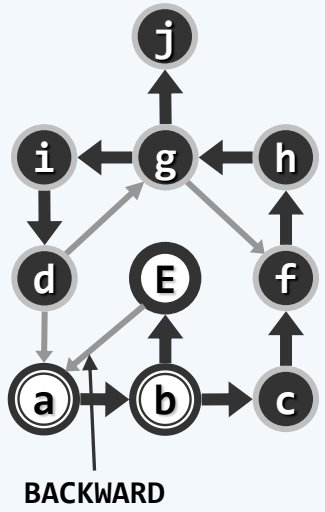
实例（无向图）



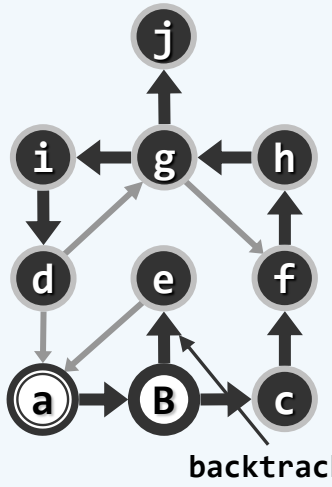
a	1	
B	2	
c	3	16
d	10	11
e		
f	4	15
g	6	13
h	5	14
i	9	12
j	7	8



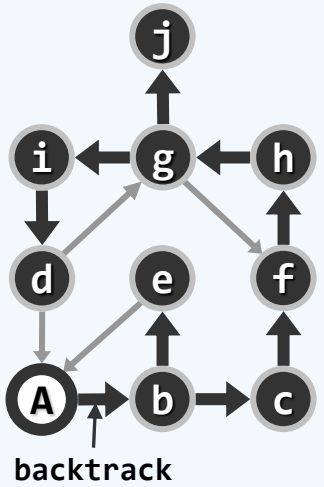
a	1	
b	2	
c	3	16
d	10	11
E	17	
f	4	15
g	6	13
h	5	14
i	9	12
j	7	8



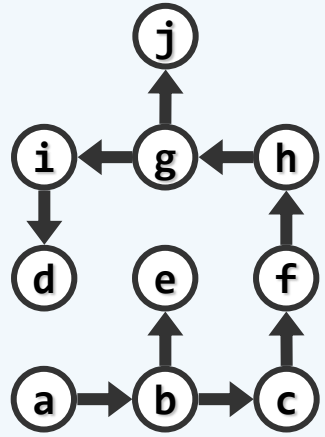
a	1	
b	2	
c	3	16
d	10	11
E	17	
f	4	15
g	6	13
h	5	14
i	9	12
j	7	8



a	1	
B	2	
c	3	16
d	10	11
e	17	18
f	4	15
g	6	13
h	5	14
i	9	12
j	7	8



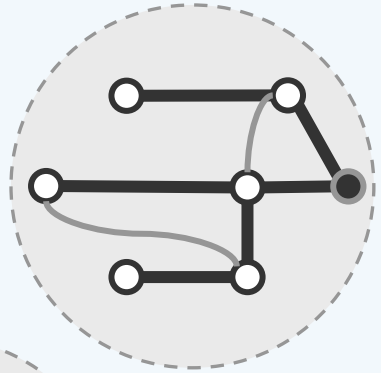
A	1	
b	2	19
c	3	16
d	10	11
e	17	18
f	4	15
g	6	13
h	5	14
i	9	12
j	7	8

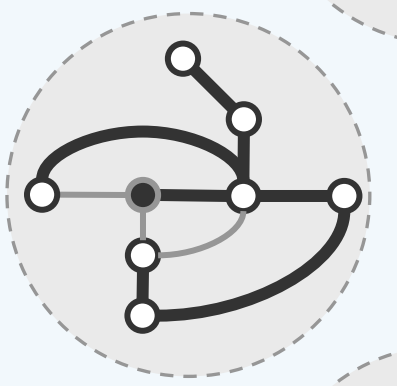


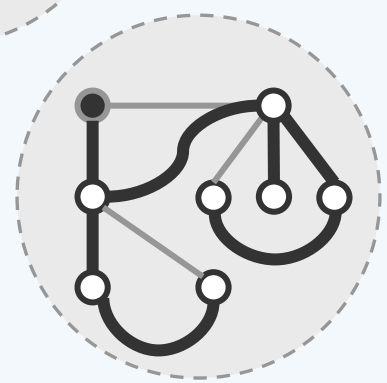
a	1	20
b	2	19
c	3	16
d	10	11
e	17	18
f	4	15
g	6	13
h	5	14
i	9	12
j	7	8

Graph::dfs()

- 与BFS(v)类似，DFS(v)也可遍历v所属分量——若含多个分量呢？
- 与bfs(s)类似（采用邻接表），dfs(s)也可在累计$O(n + e)$时间内
  对于每一连通/可达分量，从其起始顶点v进入DFS(v)恰好1次，并
  最终生成一个DFS森林（包含c棵树、n - c条树边）

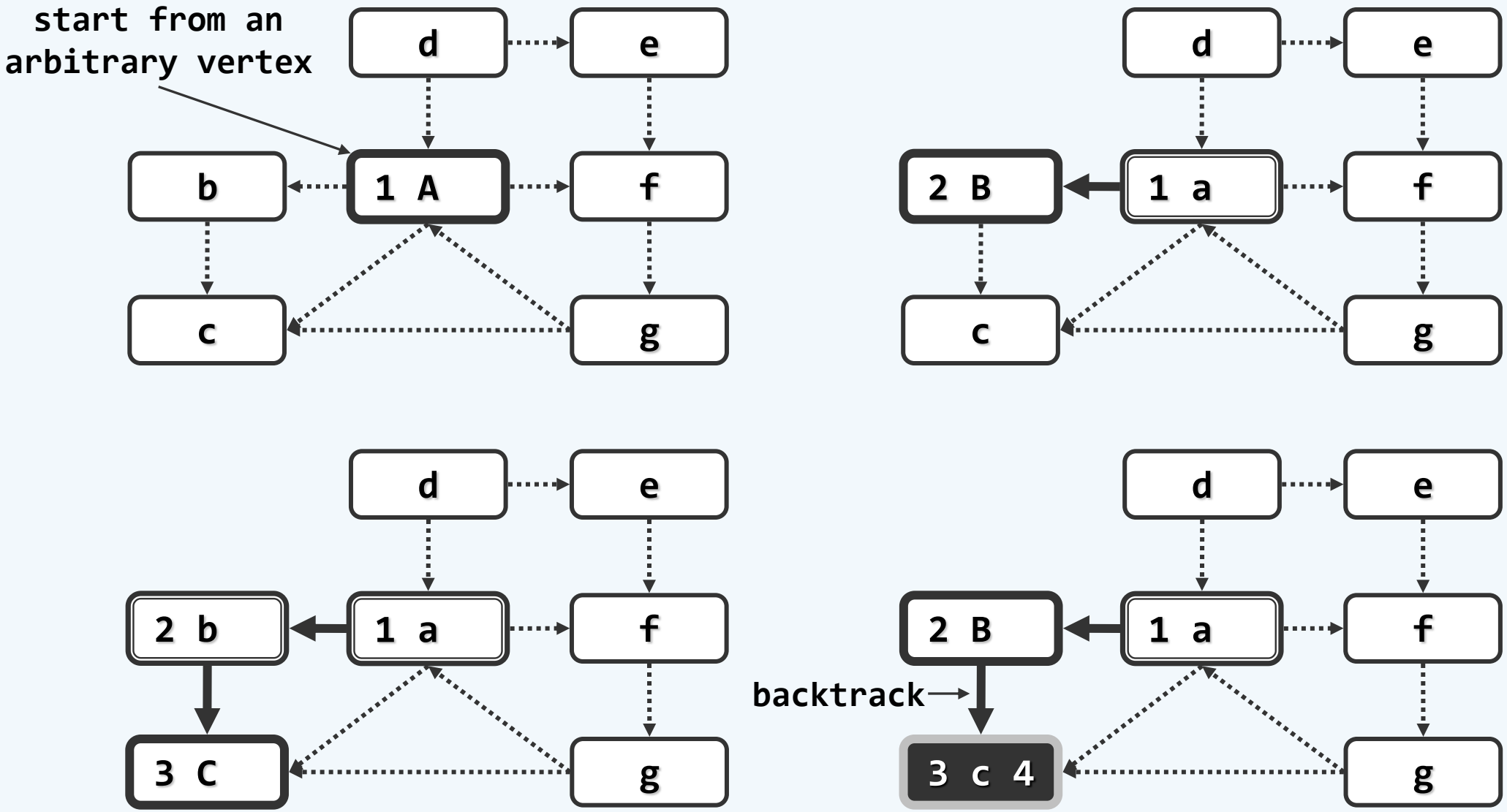




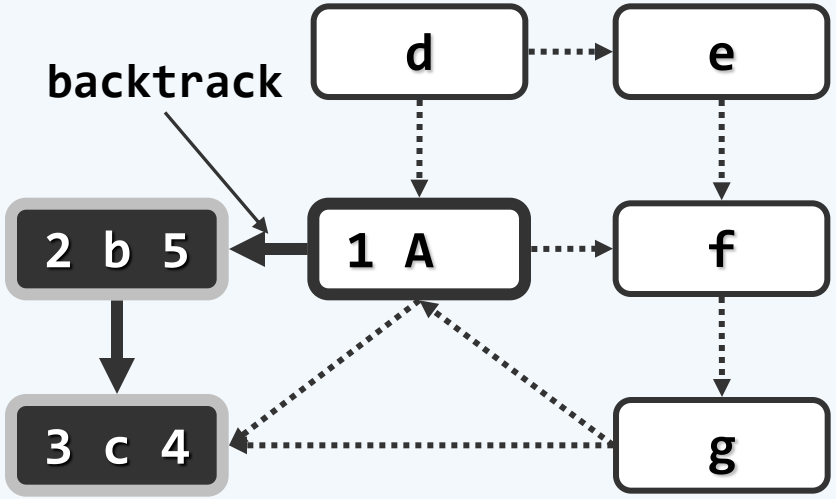


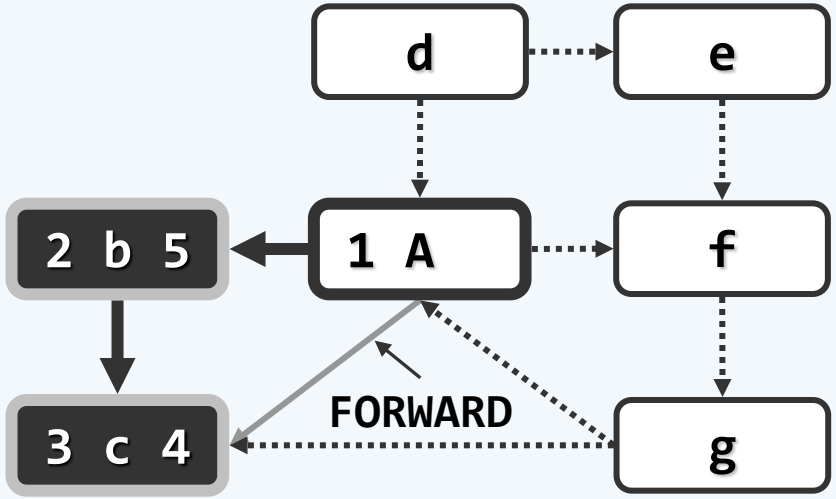
```
template <typename Tv, typename Te> //顶点类型、边类型
void Graph<Tv, Te>::dfs( int s ) { //s为起始顶点
   reset(); int clock = 0; int v = s; //初始化
   do //逐一检查所有顶点，一旦遇到尚未发现的顶点
      if ( UNDISCOVERED == status(v) )
         DFS( v, clock ); //即从该顶点出发启动一次DFS
   while ( s != ( v = ( ++v % n ) ) ); //按序号访问，故不漏不重
}
```

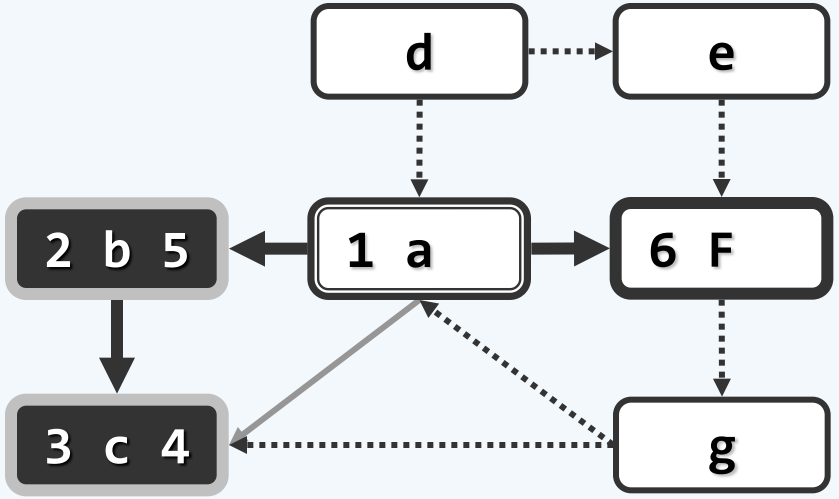
实例（有向图）

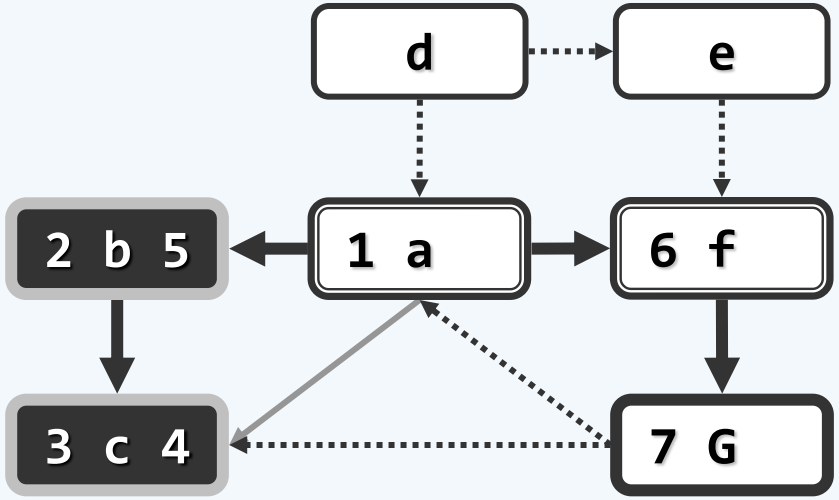

start from an arbitrary vertex
d
e
b
1 A
f
c
g
d
e
2 B
1 a
f
c
g
d
e
2 b
1 a
f
3 C
g
backtrack
d
e
2 B
1 a
f
3 c 4
g

实例（有向图）

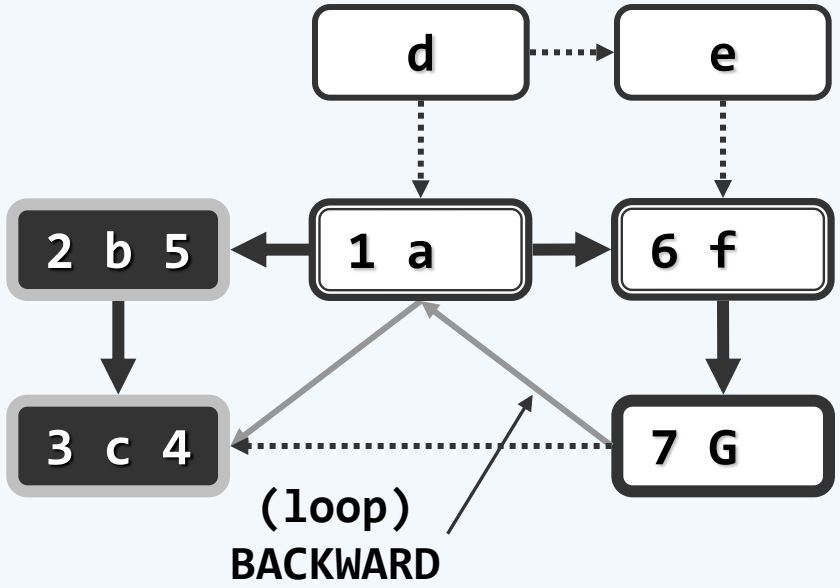

backtrack
d
e
2 b 5
1 A
f
3 c 4
g

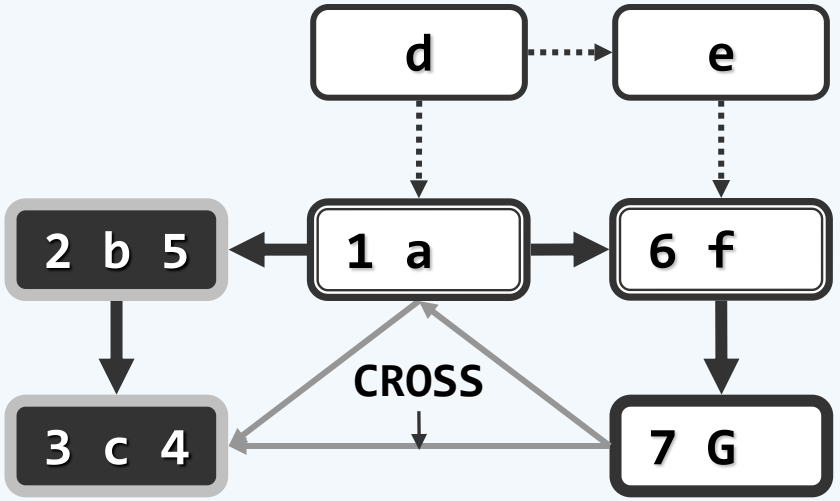

d
e
2 b 5
1 A
f
FORWARD
3 c 4
g

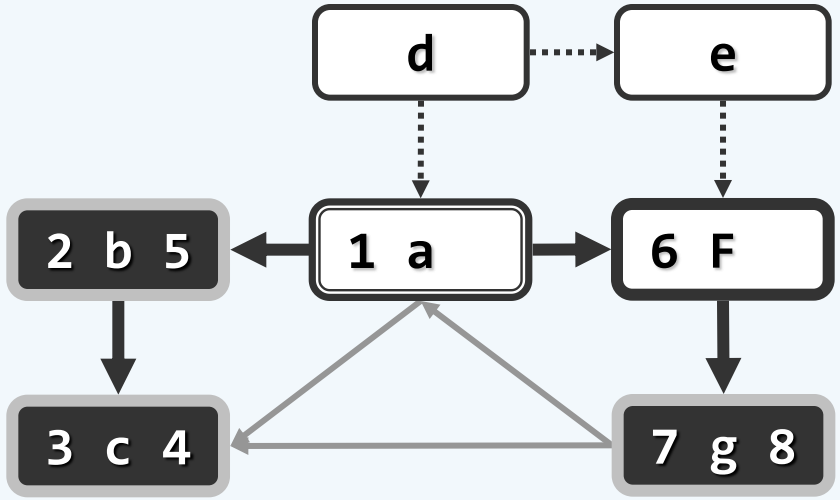

d
e
2 b 5
1 a
6 F
3 c 4
g

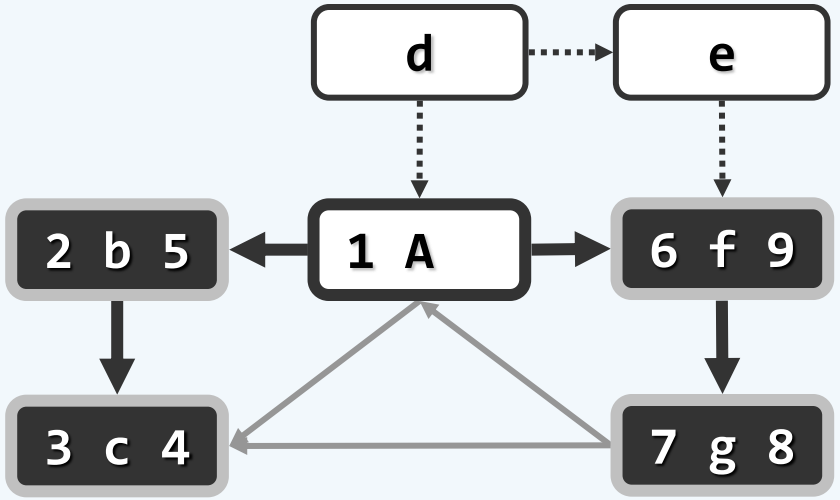

d
e
2 b 5
1 a
6 f
3 c 4
7 G

实例（有向图）

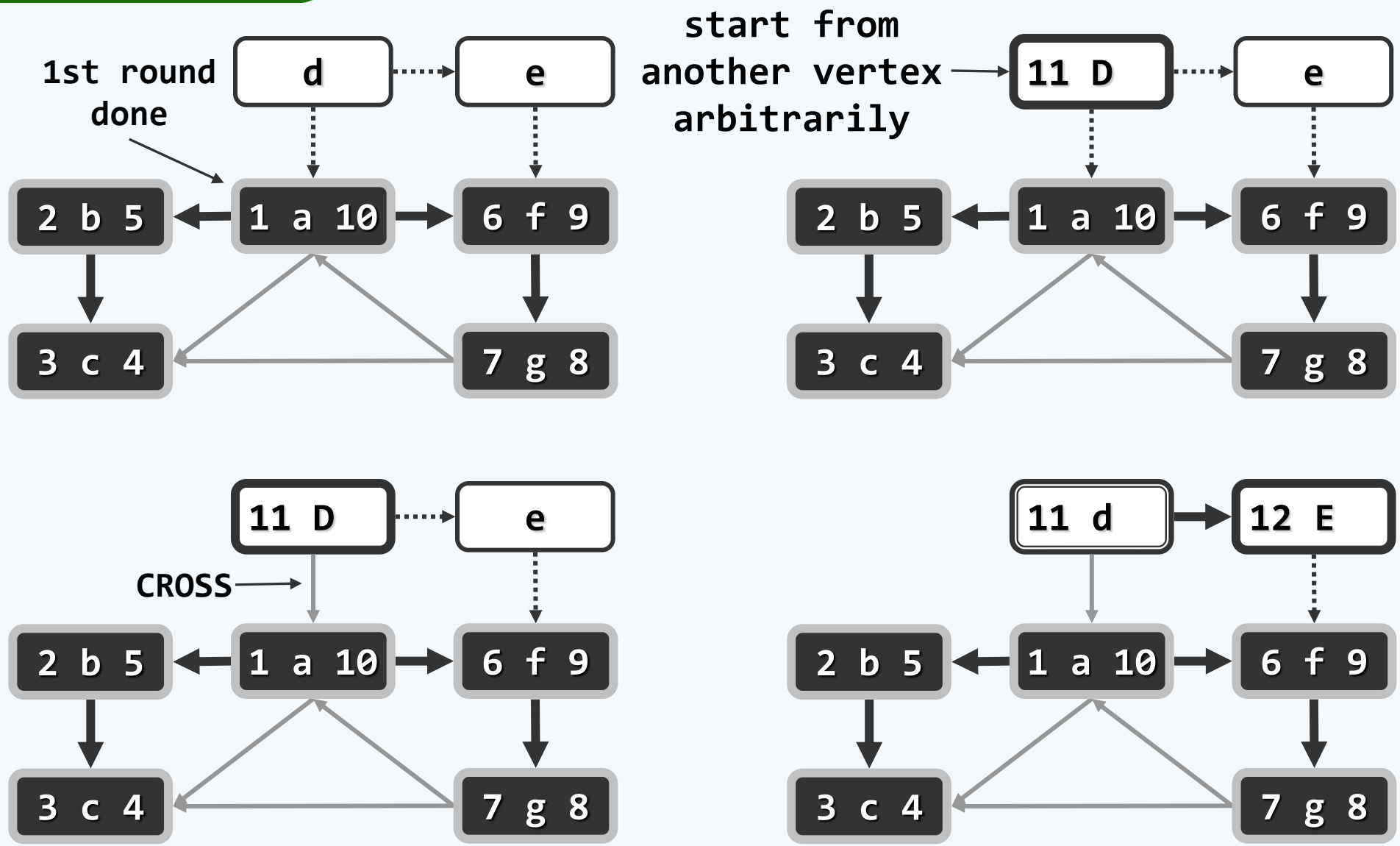

d
e
2 b 5
1 a
6 f
3 c 4
7 G
(loop)
BACKWARD


d
e
2 b 5
1 a
6 f
CROSS
3 c 4
7 G

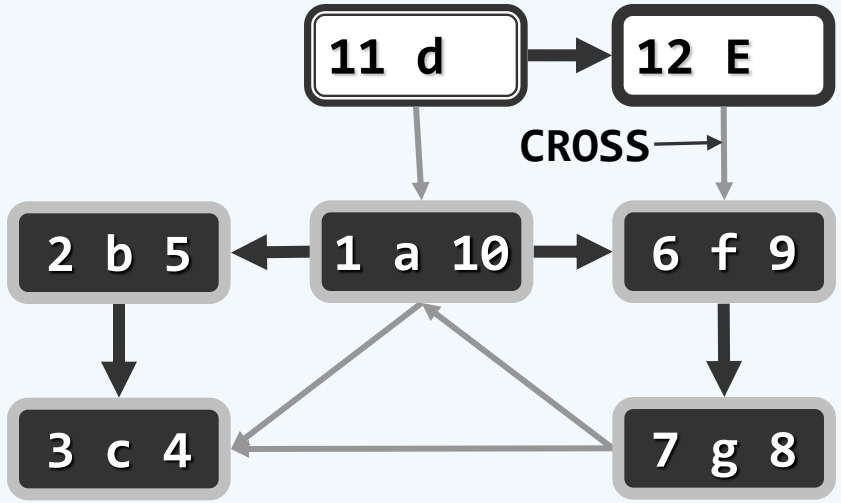

d
e
2 b 5
1 a
6 F
3 c 4
7 g 8

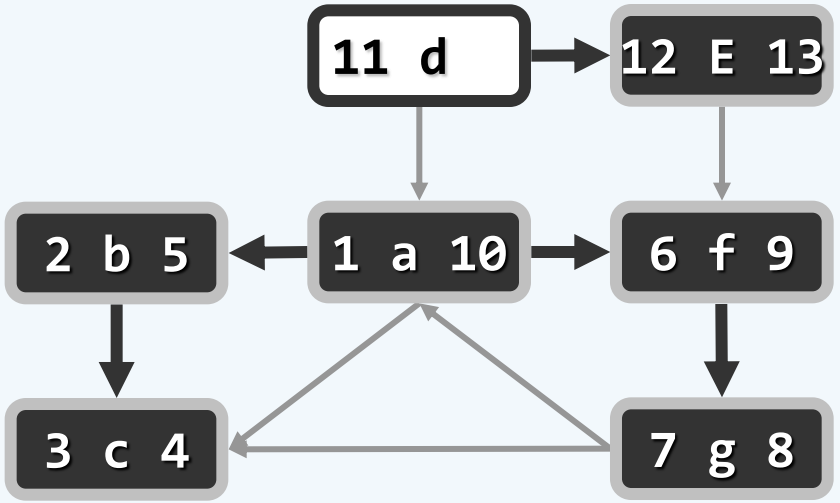

d
e
2 b 5
1 A
6 f 9
3 c 4
7 g 8

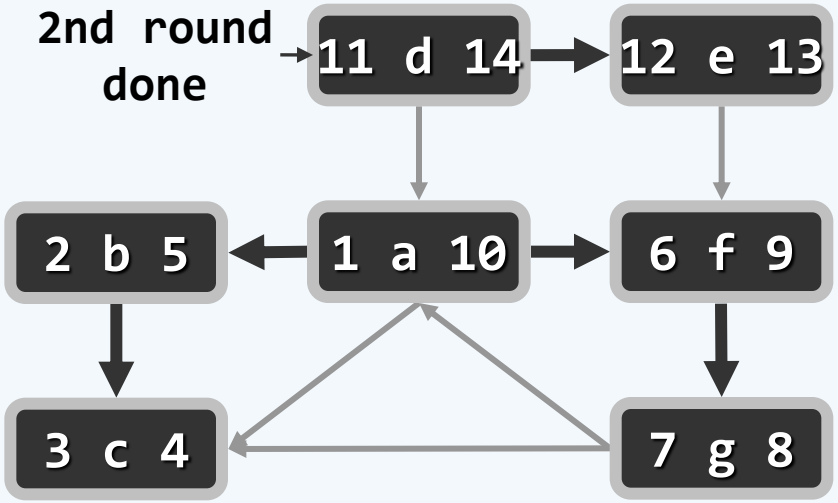
实例（有向图）

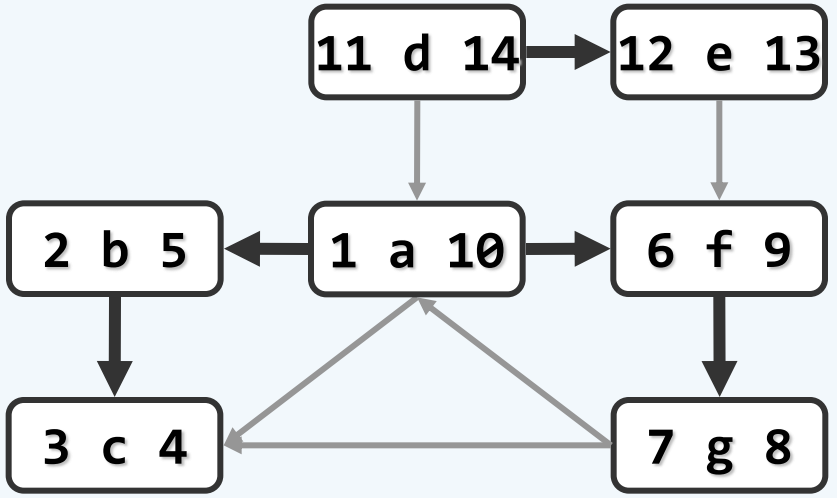

1st round done
d
e
2 b 5
1 a 10
6 f 9
3 c 4
7 g 8
start from another vertex arbitrarily
11 D
e
2 b 5
1 a 10
6 f 9
3 c 4
7 g 8
11 D
e
CROSS
2 b 5
1 a 10
6 f 9
3 c 4
7 g 8
11 d
12 E
2 b 5
1 a 10
6 f 9
3 c 4
7 g 8

实例（有向图）


11 d
12 E
CROSS
2 b 5
1 a 10
6 f 9
3 c 4
7 g 8


11 d
12 E 13
2 b 5
1 a 10
6 f 9
3 c 4
7 g 8


2nd round done
11 d 14
12 e 13
2 b 5
1 a 10
6 f 9
3 c 4
7 g 8


11 d 14
12 e 13
2 b 5
1 a 10
6 f 9
3 c 4
7 g 8

DFS树/森林

- 从顶点s出发的DFS

  在无向图中将访问与s连通的所有顶点 connected component

  在有向图中将访问由s可达的所有顶点 reachable component

- 经DFS确定的树边，不会构成回路

- 从s出发的DFS，将以s为根生成一棵DFS树；所有DFS树，进而构成DFS森林

- DFS树及森林由parent指针描述（只不过所有边取反向）

- DFS之后，我们已经知道森林乃至原图的全部信息了吗？

  就某种意义而言，是的...

括号引理

❖ **活跃期**：active[u] = (dTime[u], fTime[u])

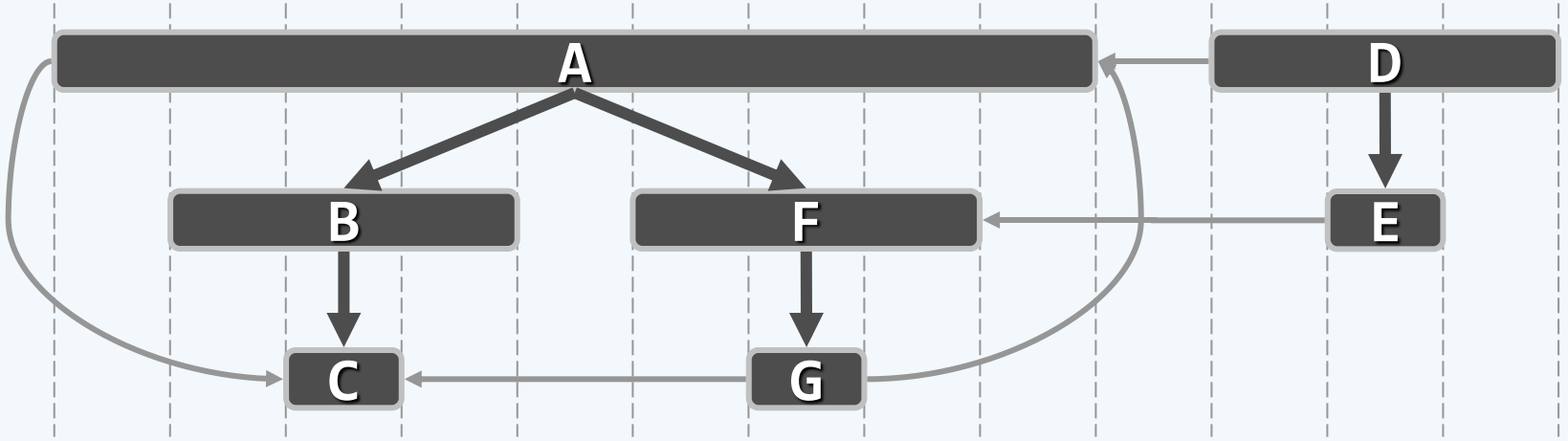
❖ Parenthesis Lemma：给定有向图G = (V, E)及其任一DFS森林，则

u是v的后代 iff active[u] ⊆ active[v]

u是v的祖先 iff active[u] ⊇ active[v]

u与v“无关” iff active[u] ∩ active[v] = ∅

❖ 仅凭status[]、dTime[]和fTime[]，即可对各边分类...



边分类

❖ TREE(v, u)： 

可从当前v进入处于UNDISCOVERED状态的u

❖ BACKWARD(v, u)： 

试图从当前v进入处于DISCOVERED状态的u

DFS发现后向边 iff 存在回路

//后向边数 == 回路数？

❖ FORWARD(v, u)： 

试图从当前顶点v进入处于VISITED状态的u，且v更早被发现

❖ CROSS(v, u)： 

试图从当前顶点v进入处于VISITED状态的u，且u更早被发现

❖ 无向图中，后向边与前向边不予区分，跨边没有

//为什么？

遍历算法的应用

连通图的支撑树（DFS/BFS Tree）	DFS/BFS
非连通图的支撑森林	DFS/BFS
连通性检测	DFS/BFS
无向图环路检测	DFS/BFS
有向图环路检测	DFS
顶点之间可达性检测/路径求解	DFS/BFS
顶点之间的最短距离	BFS
直径	BFS
Eulerian tour	DFS
拓扑排序	DFS
双连通分量、强连通分量分解	DFS
...	...