

图

概述

Le-A

但是，人的本质并不是单个人所固有的抽象物。

在其现实性上，它是一切社会关系的总和

邓俊辉

deng@tsinghua.edu.cn

# 基本术语

❖  $G = (V; E)$

vertex:  $n = |V|$



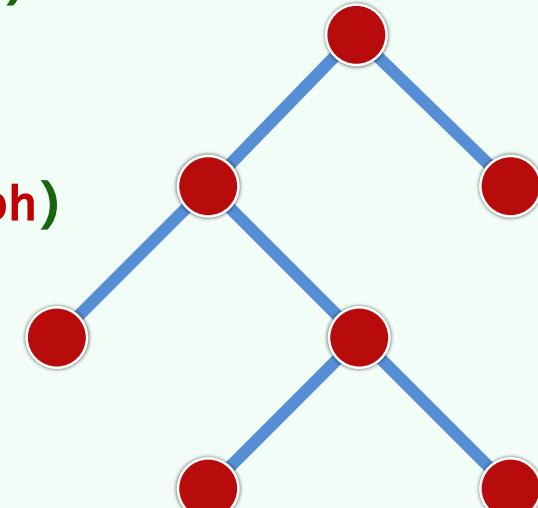
edge | arc:  $e = |E|$

❖ 同一条边的两个顶点，彼此邻接 (adjacency)

同一顶点自我邻接，构成自环 (self-loop)

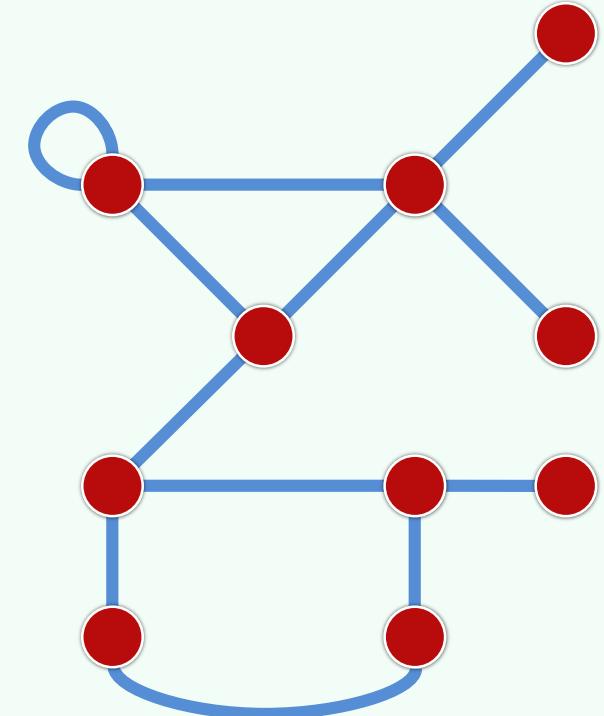
不含自环及重边，即为简单图 (simple graph)

非简单 (non-simple) 图，暂不讨论



❖ 顶点与其所属的边，彼此关联 (incidence)

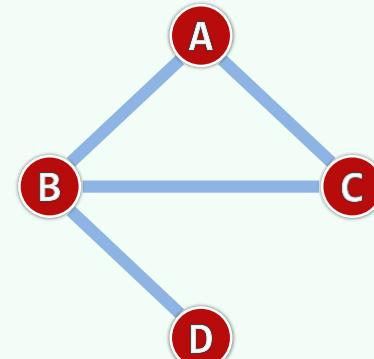
度 (degree/valency) : 与同一顶点关联的边数



# 无向图 + 有向图

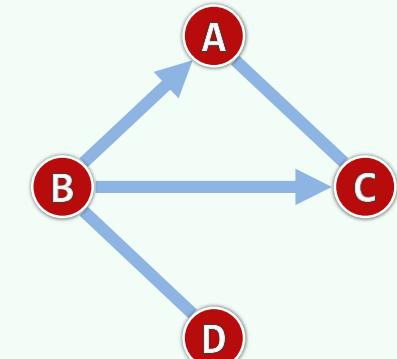
❖ 若邻接顶点 $u$ 和 $v$ 的次序无所谓

则 $(u, v)$ 为无向边 (undirected edge)



(a) undigraph

❖ 所有边均无方向的图，即无向图 (undigraph)

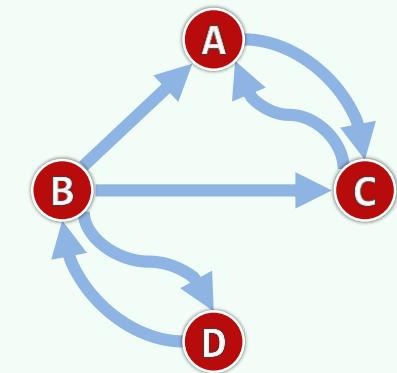


(b) mixed graph

❖ 反之，有向图 (digraph) 中均为有向边 (directed edge)

$u, v$ 分别称作边 $(u, v)$ 的尾 (tail)、头 (head)

❖ 无向边、有向边并存的图，称作混合图 (mixed graph)



(c) digraph

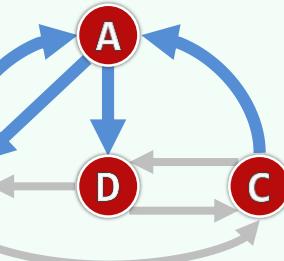
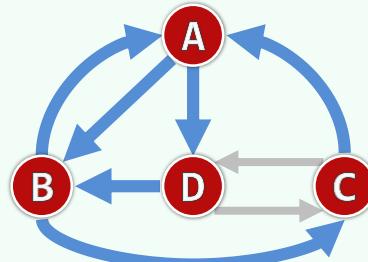
❖ 有向图通用性更强，故本章主要针对有向图介绍相关结构及算法

# 路径 + 环路

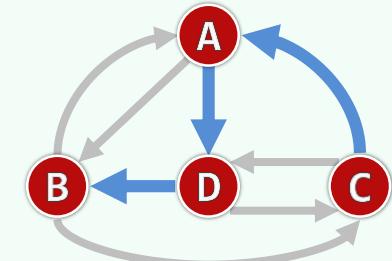
❖ 路径  $\pi = \langle v_0, v_1, \dots, v_k \rangle$

长度  $|\pi| = k$

❖ 简单路径:  $v_i \neq v_j$  除非  $i = j$



(i) path



(ii) simple path

❖ 环/环路:  $v_0 = v_k$

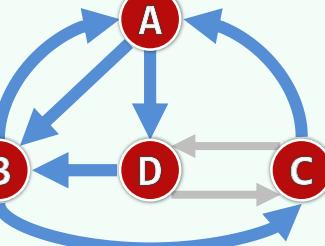
❖ 有向无环图 (DAG)

❖ 欧拉环路:  $|\pi| = |E|$

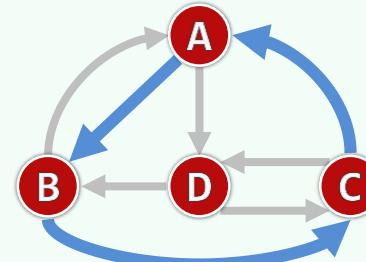
各边恰好出现一次

❖ 哈密尔顿环路:  $|\pi| = |V|$

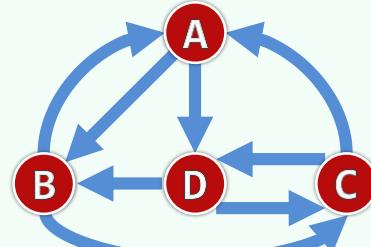
各顶点恰好出现一次



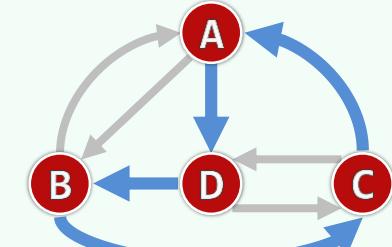
(i) cycle



(ii) simple cycle



(i) Eulerian tour

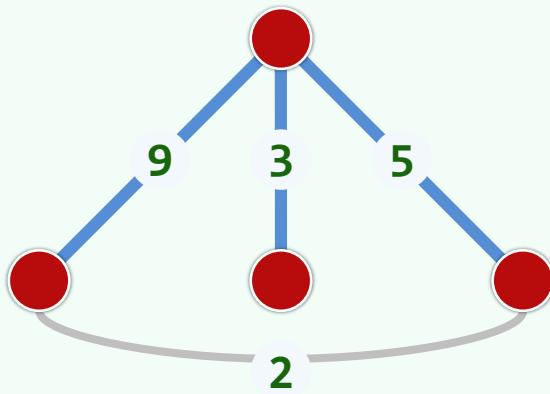


(ii) Hamiltonian tour

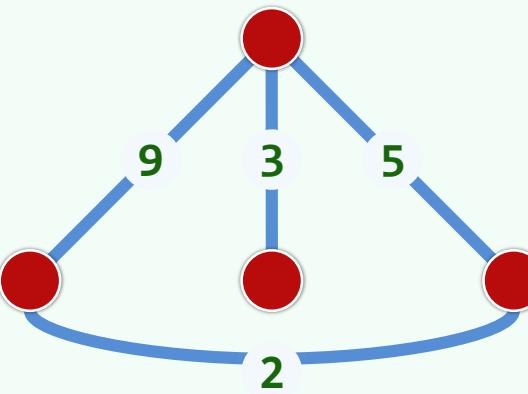
# 支撑树 + 带权网络 + 最小支撑树

❖ 图 $G = (V; E)$ 的子图 $T = (V; F)$ 若是树，即为其支撑树（spanning tree）

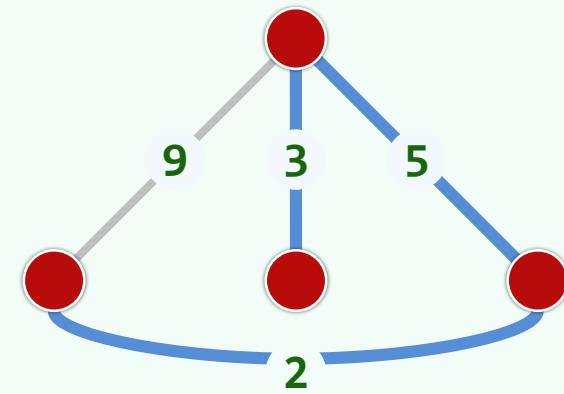
同一图的支撑树，通常并不唯一



spanning tree



weighted network  
(triangle inequality?)



minimum spanning tree

❖ 各边 $e$ 均有对应的权值 $\text{wt}(e)$ ，则为带权网络（weighted network）

❖ 同一网络的支撑树中，总权重最小者为最小支撑树（MST）

图

## 邻接矩阵：构思

10-B1

中国诗人用的“世网”二字，现在更见确切。世界真如一口网，  
横一条，竖一条，东牵西拉，把你紧紧捆扎在里面

邓俊辉  
[deng@tsinghua.edu.cn](mailto:deng@tsinghua.edu.cn)

## Graph模板类

```
template <typename Tv, typename Te> class Graph {  
private: void reset() { //所有顶点、边的辅助信息复位  
    for ( Rank v = 0; v < n; v++ ) { //顶点  
        status(v) = UNDISCOVERED; dTime(v) = fTime(v) = -1;  
        parent(v) = -1; priority(v) = INT_MAX;  
        for ( Rank u = 0; u < n; u++ ) //边  
            if ( exists(v, u) ) type(v, u) = UNDETERMINED;  
    } //for  
} //reset  
public: Rank n, e; //顶点、边数目  
/* ... 顶点操作、边操作、图算法：无论如何实现，接口必须统一 ... */  
} //Graph
```

# 邻接矩阵 + 关联矩阵

❖ adjacency matrix: 记录顶点之间的邻接关系

一一对应: 矩阵元素  $\Leftrightarrow$  图中可能存在的边

-  $A(v, u) = 1$  (若顶点v与u之间存在一条边)

-  $= 0$  (否则)

既然只考察简单图, 对角线统一设置为0

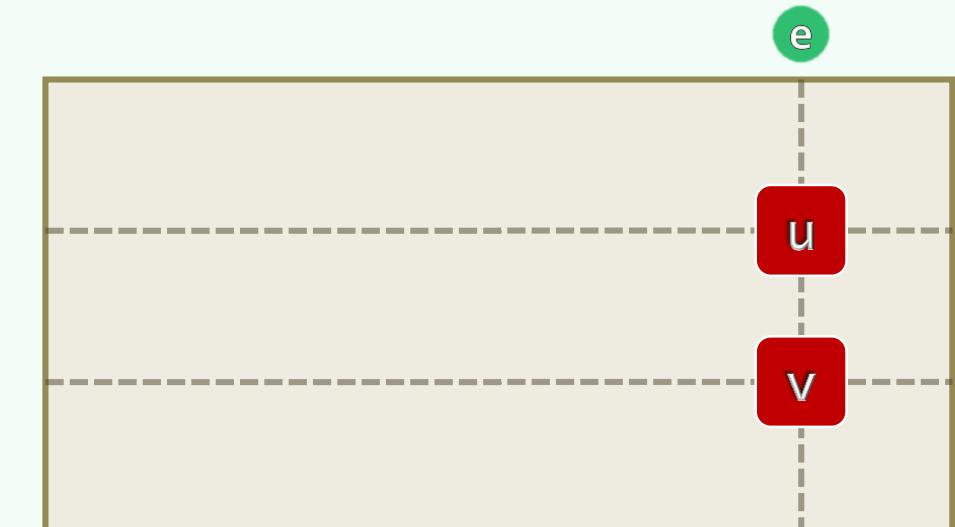
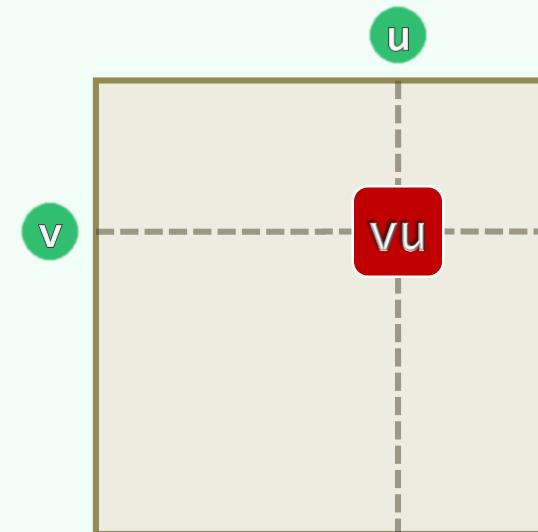
空间复杂度为  $\Theta(n^2)$ , 与图中实际的边数无关

❖ incidence matrix: 记录顶点与边之间的关联关系

空间复杂度 =  $\Theta(n \cdot e) = \mathcal{O}(n^3)$

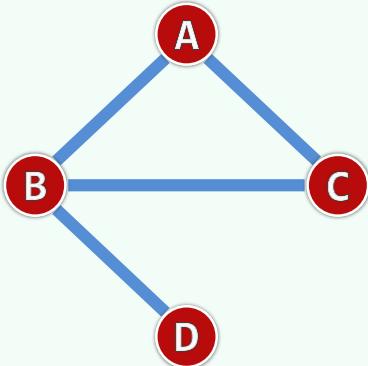
空间利用率 =  $2e/ne = 2/n \mapsto 0$

解决某些问题时十分有效

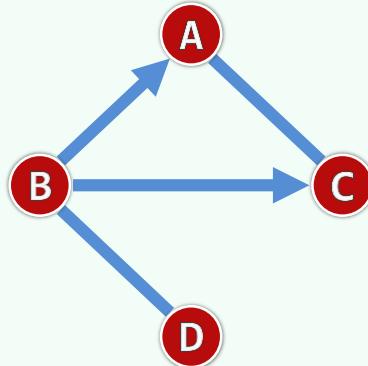


# 实例

(a) undigraph



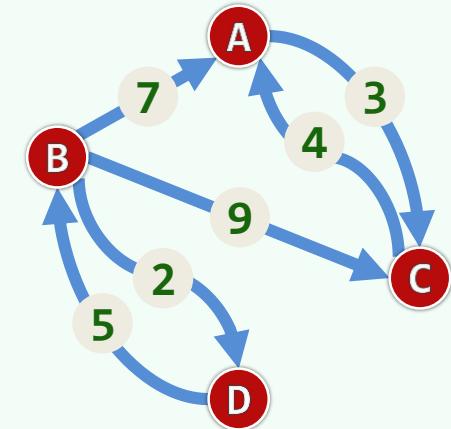
(b) digraph



redundancy

0	A	B	C	D
A		1	1	
B	1		1	1
C	1	1		
D	1			

(c) network



0	A	B	C	D
A			1	
B	1		1	1
C	1			
D		1		

图

## 邻接矩阵：模板实现

10-B2

你拿了这图，到临安府找一家客店或是寺观住下，三月之后，  
我派人前来取回。图中一切，只许心记，不得另行抄录印摹

邓俊辉

deng@tsinghua.edu.cn

## Vertex

```
using VStatus = enum { UNDISCOVERED, DISCOVERED, VISITED };

template <typename Tv> struct Vertex { //不再严格封装

    Tv data; int inDegree, outDegree;

    VStatus status; // (如上三种) 状态

    int dTime, fTime; //时间标签

    Rank parent; //在遍历树中的父节点

    int priority; //在遍历树中的优先级 (最短通路、极短跨边等)

    Vertex( Tv const & d ) : //构造新顶点

        data( d ), inDegree( 0 ), outDegree( 0 ), status( UNDISCOVERED ),

        dTime( -1 ), fTime( -1 ), parent( -1 ), priority( INT_MAX ) {}

};
```

# Edge

```
using EType = enum { UNDETERMINED, TREE, CROSS, FORWARD, BACKWARD };

template <typename Te> struct Edge { //不再严格封装

    Te data; //数据

    int weight; //权重

    EType type; //在遍历树中所属的类型

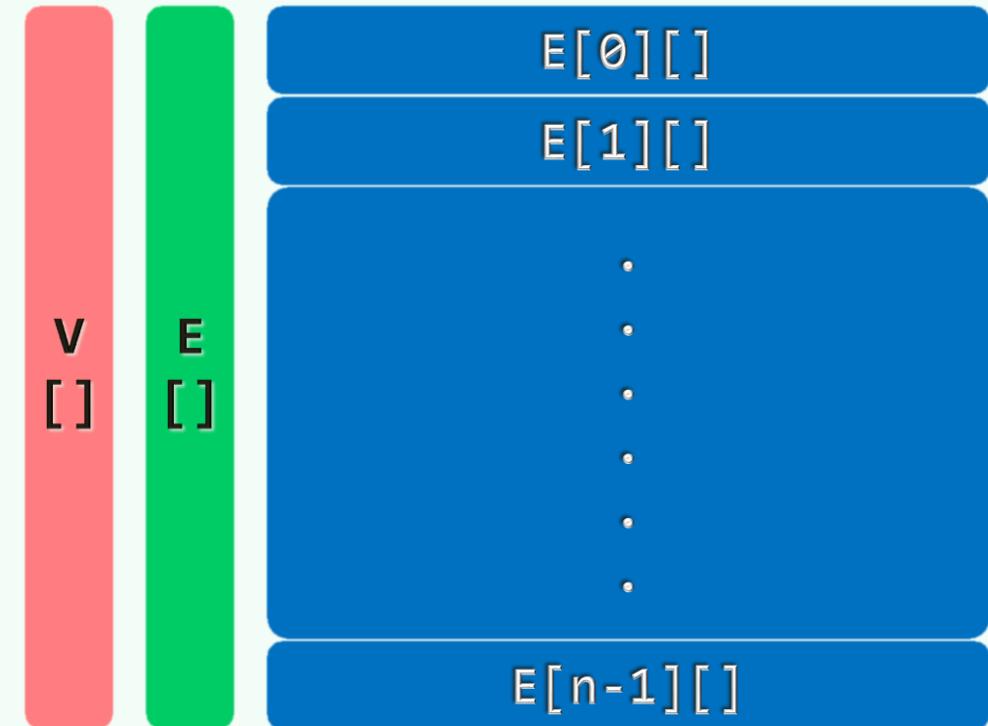
    Edge( Te const & d, int w ) : //构造新边

        data(d), weight(w), type(UNDETERMINED) {}

};
```

# GraphMatrix

```
template <typename Tv, typename Te> class GraphMatrix : public Graphprivate:  
    Vector< Vertex<Tv> > V; //顶点集  
    Vector< Vector< Edge<Te>*> > E; //边集  
public: // 操作接口: 顶点相关、边相关、...  
    GraphMatrix() { n = e = 0; } //构造  
    ~GraphMatrix() { //析构  
        for ( Rank v = 0; v < n; v++ )  
            for ( Rank u = 0; u < n; u++ )  
                delete E[v][u]; //清除所有边记录  
    }  
};
```



图

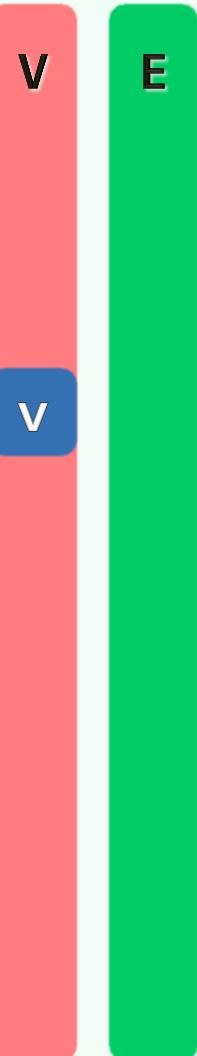
邻接矩阵：静态操作

10-B3

邓俊辉

deng@tsinghua.edu.cn

# 顶点的读写



```
Tv & vertex(Rank v) { return V[v].data; } //数据  
int inDegree(Rank v) { return V[v].inDegree; } //入度  
int outDegree(Rank v) { return V[v].outDegree; } //出度  
VStatus & status(Rank v) { return V[v].status; } //状态  
int & dTime(Rank v) { return V[v].dTime; } //时间标签dTime  
int & fTime(Rank v) { return V[v].fTime; } //时间标签fTime  
Rank & parent(Rank v) { return V[v].parent; } //在遍历树中的父亲  
int & priority(Rank v) { return V[v].priority; } //优先级数
```

# 边的读写

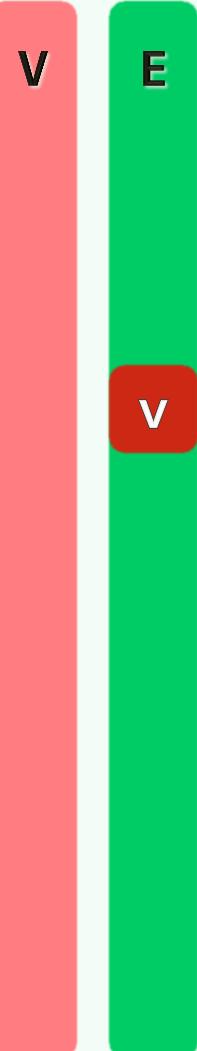


```
bool exists( Rank v, Rank u ) { //判断边(v, u)是否存在 (短路求值)  
    return (v < n) && (u < n) && (E[v][u] != NULL);  
}  
//以下假定exists(v, u) = true
```



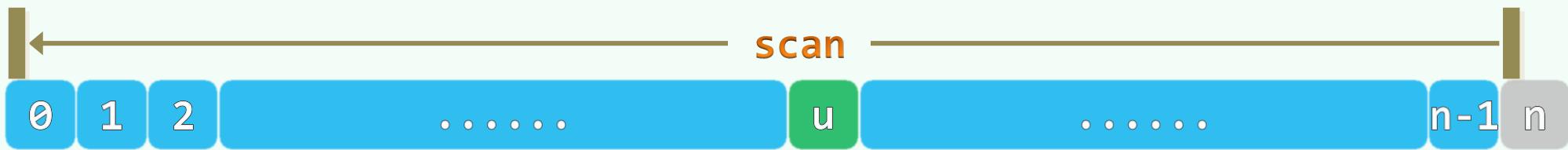
```
Te & edge( Rank v, Rank u ) { return E[v][u]->data; } //数值  
EType & type( Rank v, Rank u ) { return E[v][u]->type; } //类型  
int & weight( Rank v, Rank u ) { return E[v][u]->weight; } //权重
```

# 邻点的枚举



对于任意顶点v，如何枚举其所有的邻接顶点 (neighbor) ?

Rank firstNbr( Rank v ) { return nextNbr( v, n ); } //假想哨兵



Rank nextNbr( Rank v, Rank u ) { //若已枚举至邻居u，则转向下一邻居

while ( ( -1 != --u ) && ! exists( v, u ) ); //逆向顺序查找

return u;

} // $\mathcal{O}(n)$ ——改用邻接表，可提高至 $\mathcal{O}(1 + \text{outDegree}(v))$

图

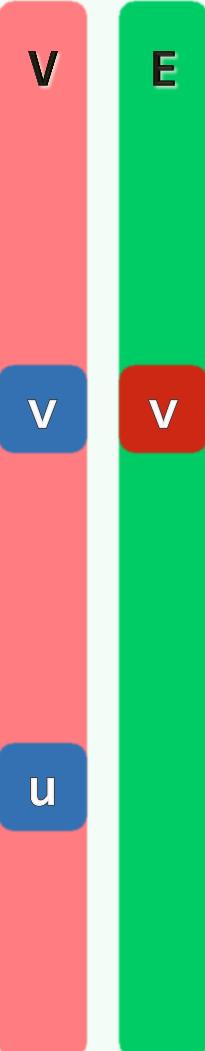
## 邻接矩阵：动态操作

10-B4

邓俊辉

deng@tsinghua.edu.cn

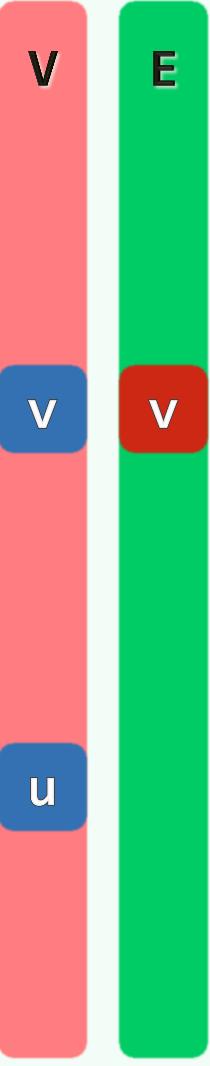
# 边的插入



```
void insert( Te const & edge, int w, Rank v, Rank u ) {  
    if ( exists(v, u) ) return; //忽略已有的边  
    E[v][u] = new Edge<Te>( edge, w ); //创建新边 (权重为w)  
    e++; //更新边计数  
    v[v].outDegree++; //更新顶点v的出度  
    v[u].inDegree++; //更新顶点u的入度  
}
```



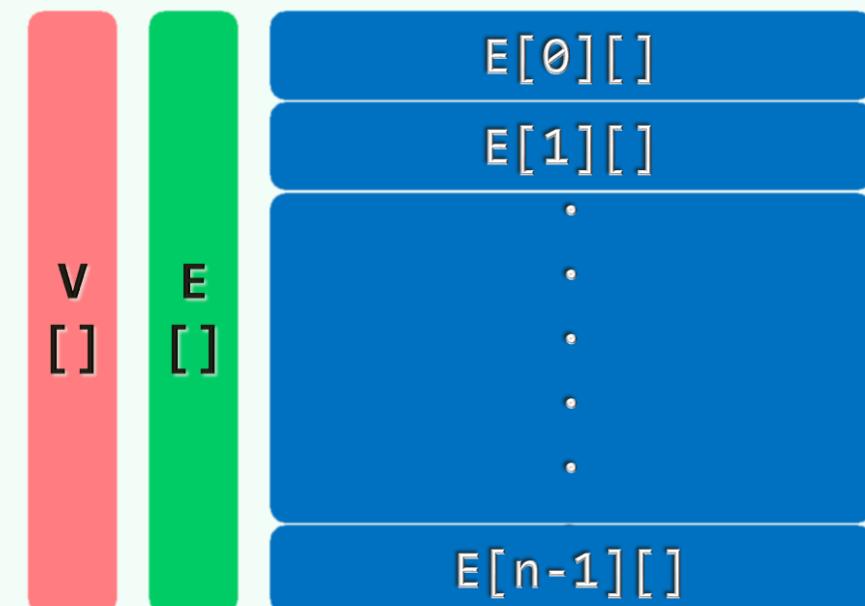
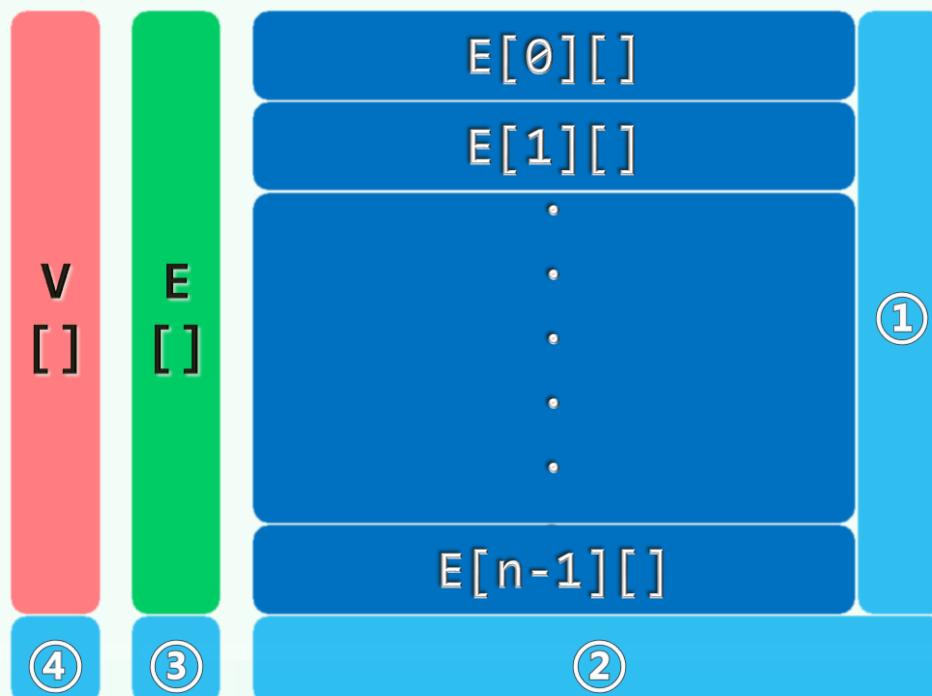
# 边的删除



```
Te remove( Rank v, Rank u ) { //删除 (已确认存在的) 边(v, u)  
    Te eBak = edge(v, u); //备份边(v, u)的信息  
  
    delete E[v][u]; E[v][u] = NULL; //删除边(v, u)  
  
    e--; //更新边计数  
  
    v[v].outDegree--; //更新顶点v的出度  
  
    v[u].inDegree--; //更新顶点u的入度  
  
    return eBak; //返回被删除边的信息  
}
```

# 顶点插入

```
Rank insert( Tv const & vertex ) { //插入顶点, 返回编号  
    for ( Rank u = 0; u < n; u++ ) E[u].insert( NULL ); n++; //①  
    E.insert( Vector< Edge<Te>*>( n, n, NULL ) ); //②③  
    return V.insert( Vertex<Tv>( vertex ) ); //④  
}
```



# 顶点删除

```
Tv remove( Rank v ) { //删除顶点及其关联边，返回该顶点信息  
    for ( Rank u = 0; u < n; u++ ) //删除所有出边  
        if ( exists( v, u ) ) { delete E[v][u]; V[u].inDegree--; e-- }  
    E.remove(v); n--; //删除第v行  
    Tv vBak = vertex( v ); V.remove( v ); //备份之后，删除顶点v  
    for ( Rank u = 0; u < n; u++ ) //删除所有入边及第v列  
        if ( Edge<Te> * x = E[u].remove( v ) )  
            { delete x; V[u].outDegree--; e--; }  
    return vBak; //返回被删除顶点的信息  
}
```

# 邻接矩阵：性能分析

图

10-B5

邓俊辉

deng@tsinghua.edu.cn

# 优点

❖ 直观，易于理解和实现

❖ 适用范围广泛

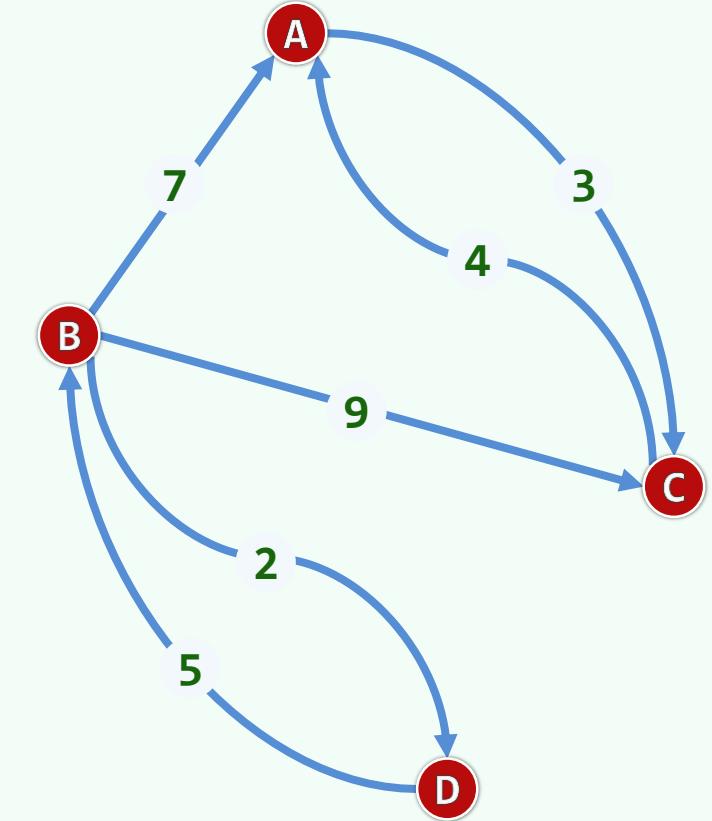
尤其适用于稠密图 (dense graph)

❖ 判断两点之间是否存在联边： $\mathcal{O}(1)$

❖ 获取顶点的（出/入）度数： $\mathcal{O}(1)$

添加、删除边后更新度数： $\mathcal{O}(1)$

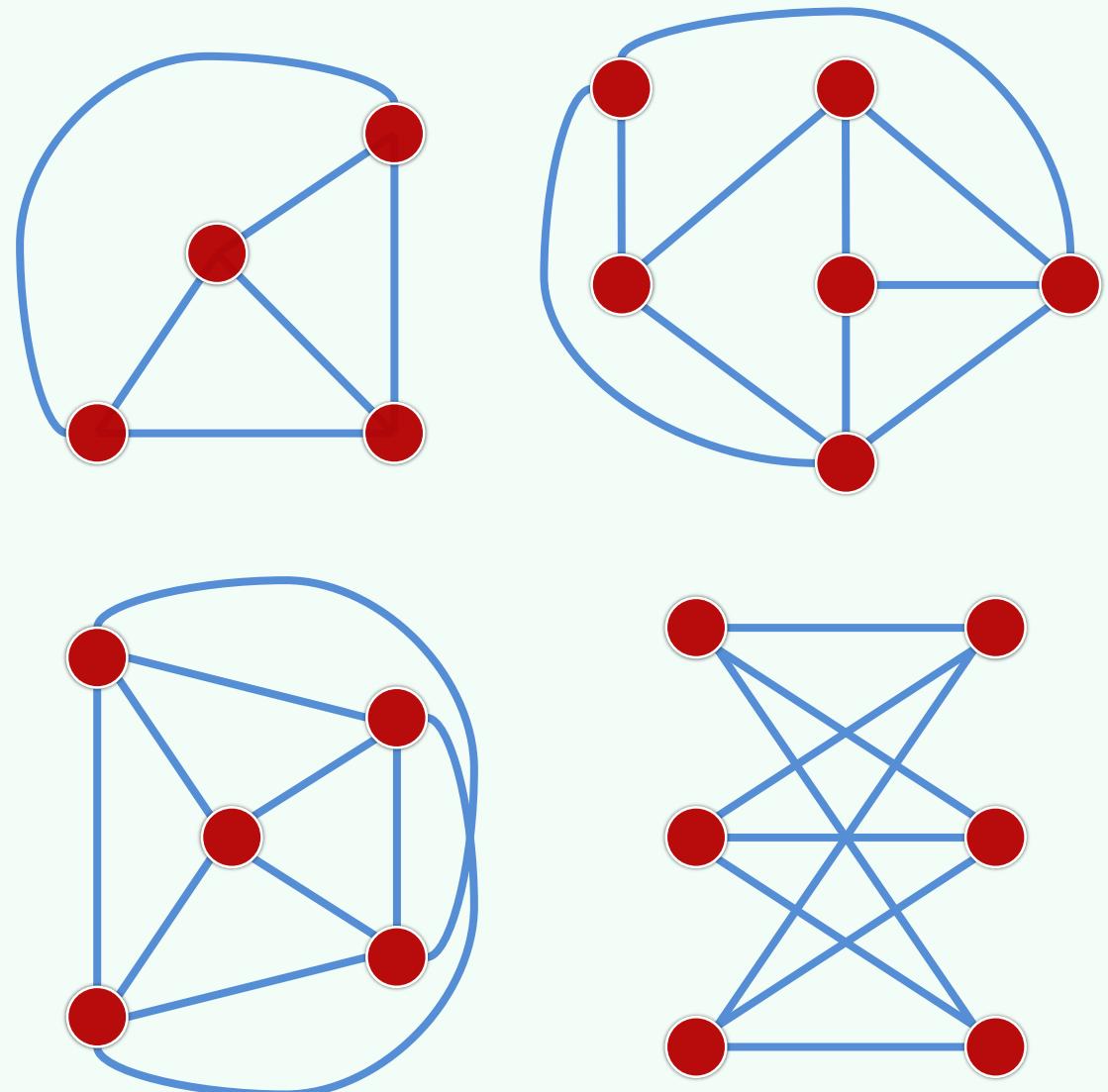
$\infty$	A	B	C	D
A			3	
B	7		9	2
C	4			
D		5		



❖ 扩展性 (scalability)：得益于Vector良好的控制策略，空间溢出等情况可被“透明地”处理

# 缺点

- ❖  $\Theta(n^2)$  空间，与边数无关！
- ❖ 真会有这么多条边吗？不妨考察一类特定的图...
- ❖ 平面图 (planar graph) : 可嵌入于平面的图
- ❖ Euler's formula (1750):  
 $v - e + f - c = 1$ , for any PG
- ❖ 平面图:  $e \leq 3n - 6 = \mathcal{O}(n) \ll n^2$   
此时, 空间利用率  $\approx 1/n$
- ❖ 稀疏图 (sparse graph)  
空间利用率同样很低, 可采用压缩存储技术



Le-C

图

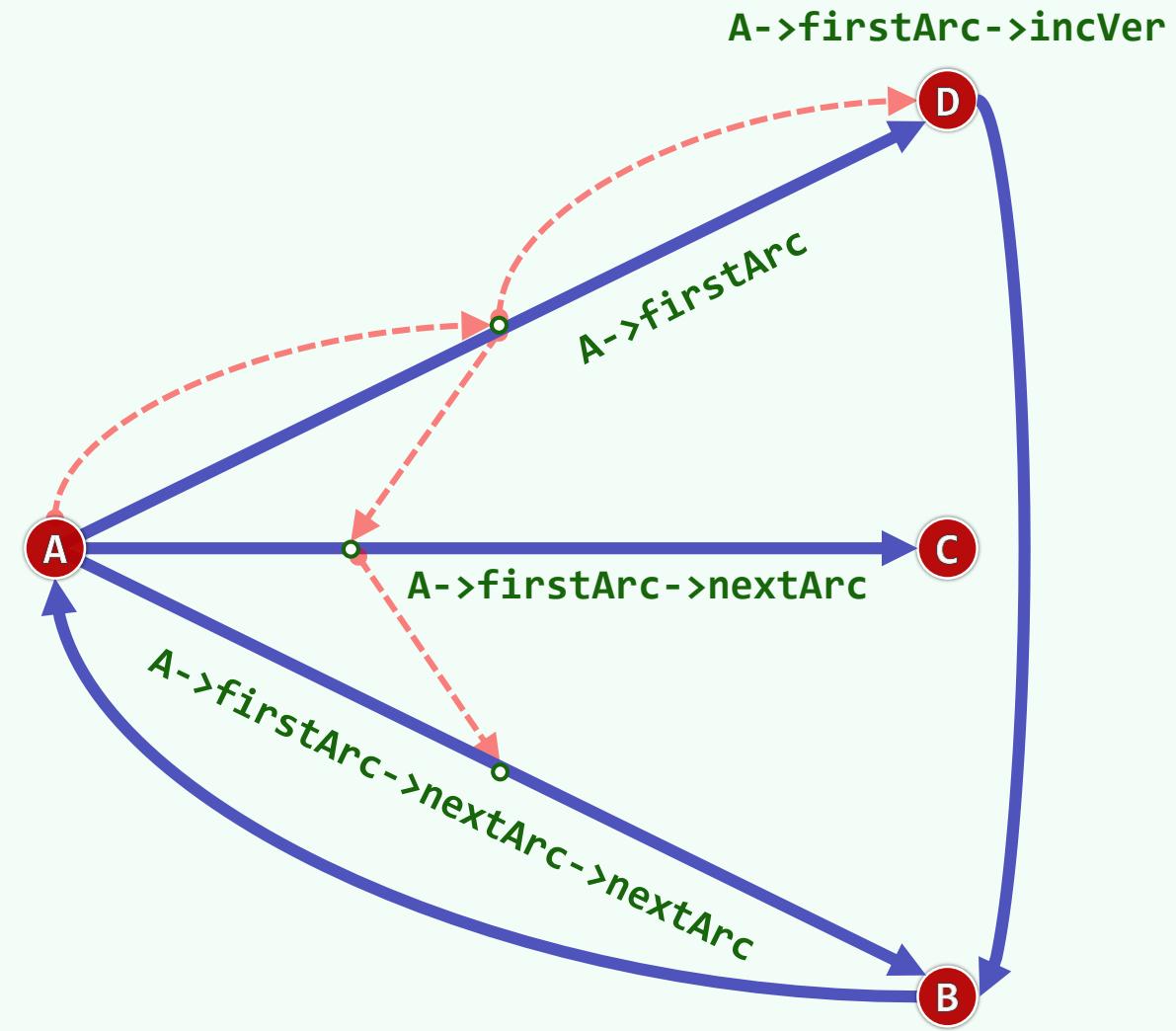
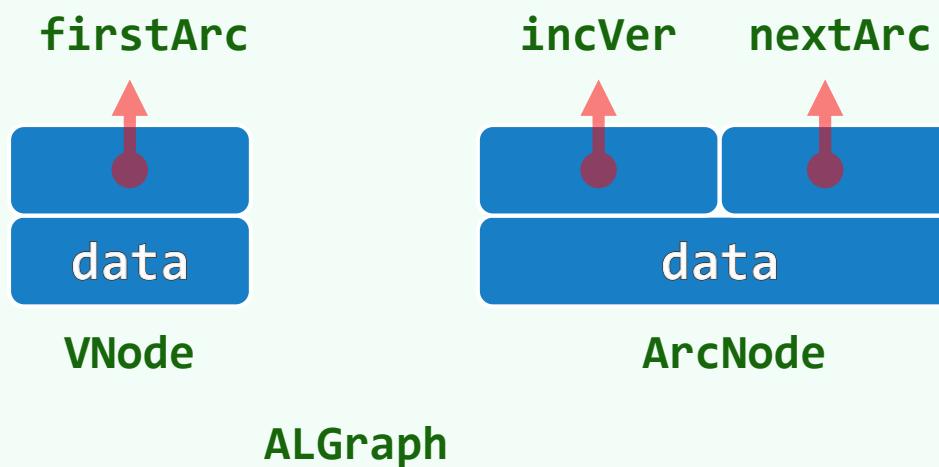
邻接表

邓俊辉

deng@tsinghua.edu.cn

# 邻接表

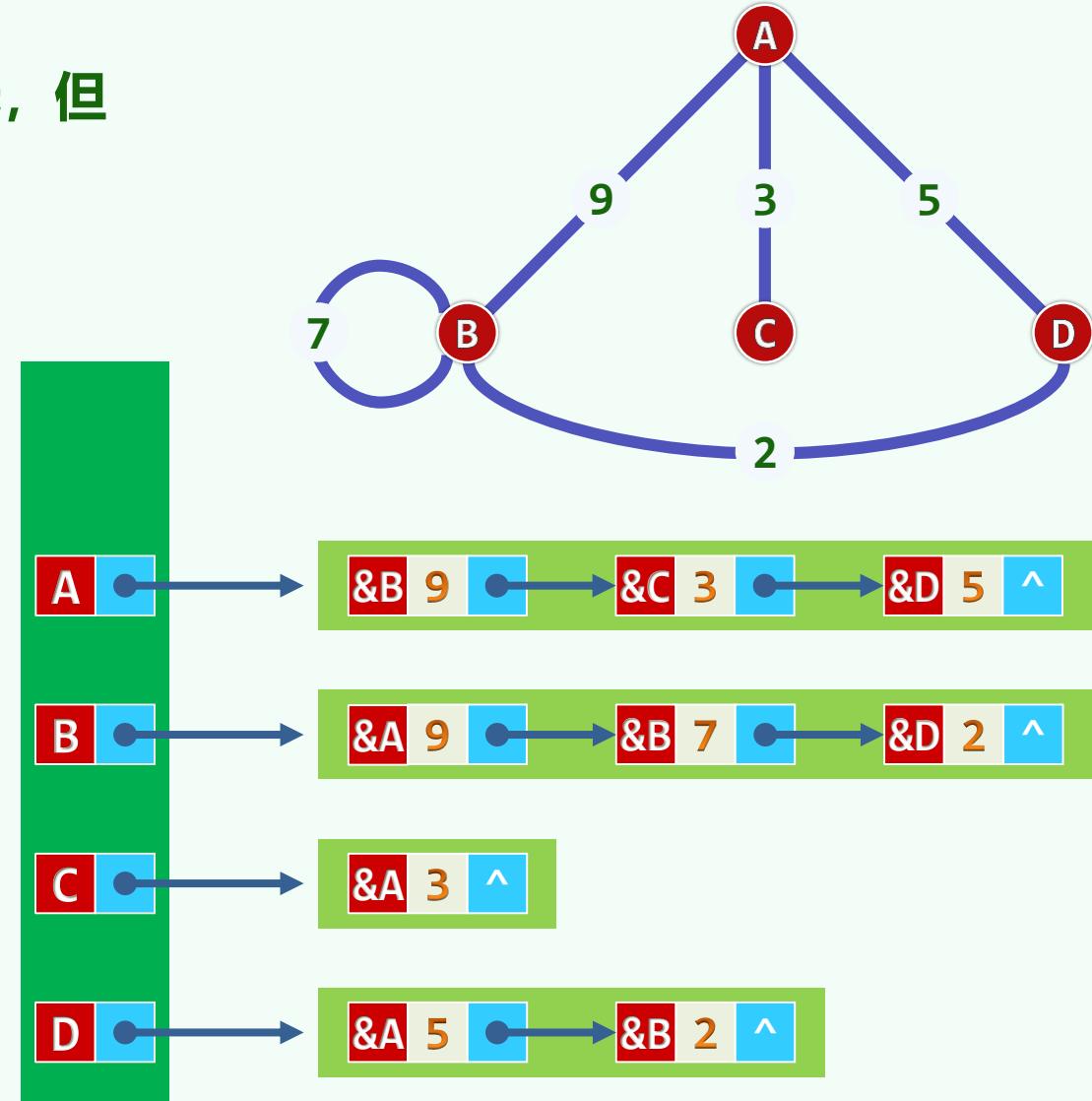
- ❖ 如何避免邻接矩阵的空间浪费?
- ❖ 将邻接矩阵的各行组织为列表，只记录存在的边
- ❖ 等效于，每一顶点v对应于列表：  
 $L_v = \{ u \mid \langle v, u \rangle \in E \}$



## 实例

❖ 4个顶点，5条弧：不必占用 $4 \times 4 = 16$ 个单元，但  
还是占用了9个单元，另加4个表头

$\infty$	A	B	C	D
A		9	3	5
B	9	7		2
C	3			
D	5	2		



# 空间复杂度

❖ 有向图 =  $\Theta(n + e)$

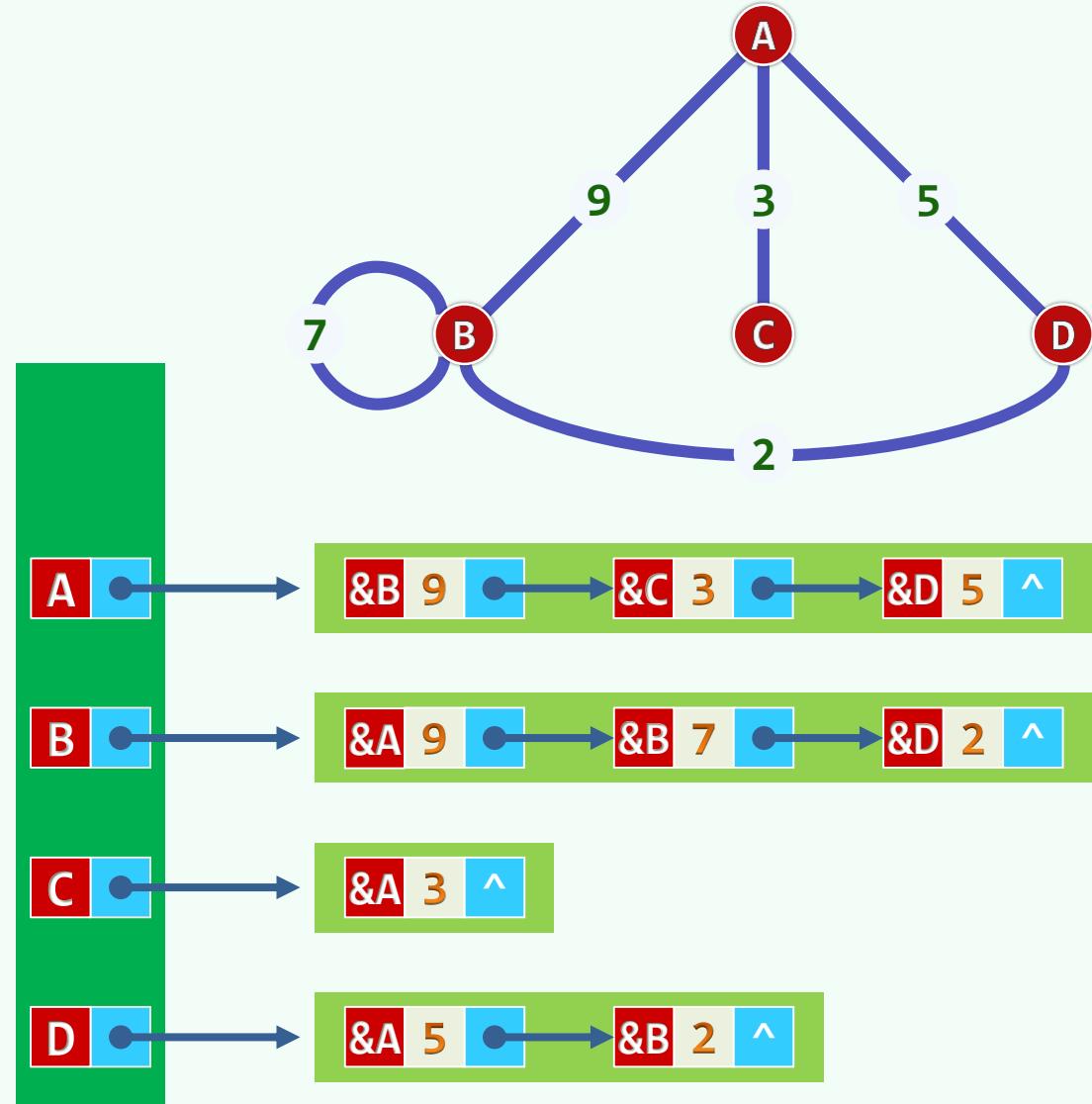
❖ 无向图 =  $\Theta(n + 2 \times e) = \Theta(n + e)$

- 注意：无向弧被重复存储
- 问题：如何改进？

❖ 适用于稀疏图

❖ 平面图 =  $\Theta(n + 3 \times n) = \Theta(n)$

较之邻接矩阵，有极大改进



## 时间复杂度 (1/2)

- ❖ 建立邻接表 (递增式构造) :  $\mathcal{O}(n + e)$  //如何实现
- ❖ 枚举所有以顶点v为尾的弧:  $\mathcal{O}(1 + \deg(v))$  //遍历v的邻接表
- ❖ 枚举 (无向图中) 顶点v的邻居:  $\mathcal{O}(1 + \deg(v))$  //遍历v的邻接表
- ❖ 枚举所有以顶点v为头的弧:  $\mathcal{O}(n + e)$  //遍历所有邻接表  
可改进至  $\mathcal{O}(1 + \deg(v))$  //建立逆邻接表——为此, 空间需增加多少?
- ❖ 计算顶点v的出度/入度:
  - 增加度数记录域:  $\mathcal{O}(n)$ 附加空间
  - 增加/删除弧时更新度数:  $\mathcal{O}(1)$ 时间 //总体  $\mathcal{O}(e)$ 时间
  - 每次查询:  $\mathcal{O}(1)$ 时间!

## 时间复杂度 (2/2)

❖ 给定顶点 $u$ 和 $v$ , 判断是否 $\langle u, v \rangle \in E$

- 有向图: 搜索 $u$ 的邻接表,  $\mathcal{O}(\deg(u))$
- 无向图: 搜索 $u$ 或 $v$ 的邻接表,  $\mathcal{O}(\max(\deg(u), \deg(v)))$
- “并行” 搜索:  $\mathcal{O}(\min(\deg(u), \deg(v)))$

能够达到邻接矩阵的 $\mathcal{O}(1)$ 吗?

❖ 散列! 如果装填因子选取得当 //保持兴趣

- 弧的判定: expected- $\mathcal{O}(1)$ , 与邻接矩阵“相同”
- 空间:  $\mathcal{O}(n + e)$ , 与邻接表相同

❖ 为何有时仍使用邻接矩阵? 仅仅因为实现简单? 不, 有更多用处! 比如, 可处理 Euclidean graph 和 intersection graph 之类的隐式图 (implicitly-represented graphs)

# 取舍原则

❖ 空间/速度

❖ 顶点类型

- bit
- int
- float
- struct
- class
- ...

❖ 弧类型 (方向 / 权值)

❖ 图类型 (稀疏 / 稠密)

适用场合

邻接矩阵

邻接表

经常检测边的存在

经常计算顶点的度数

经常做边的插入/删除

顶点数目不确定

图的规模固定

经常做遍历

稠密图

稀疏图

图

## 广度优先搜索：算法

10.-D1

邓俊辉

deng@tsinghua.edu.cn

我放下书，想，这么大一座园子，要在其中找到她的儿子，母亲走过了多少焦灼的路

# Breadth-First Search

❖ 始自顶点s的广度优先搜索

访问顶点s

依次访问s所有尚未访问的邻接顶点

依次访问它们尚未访问的邻接顶点

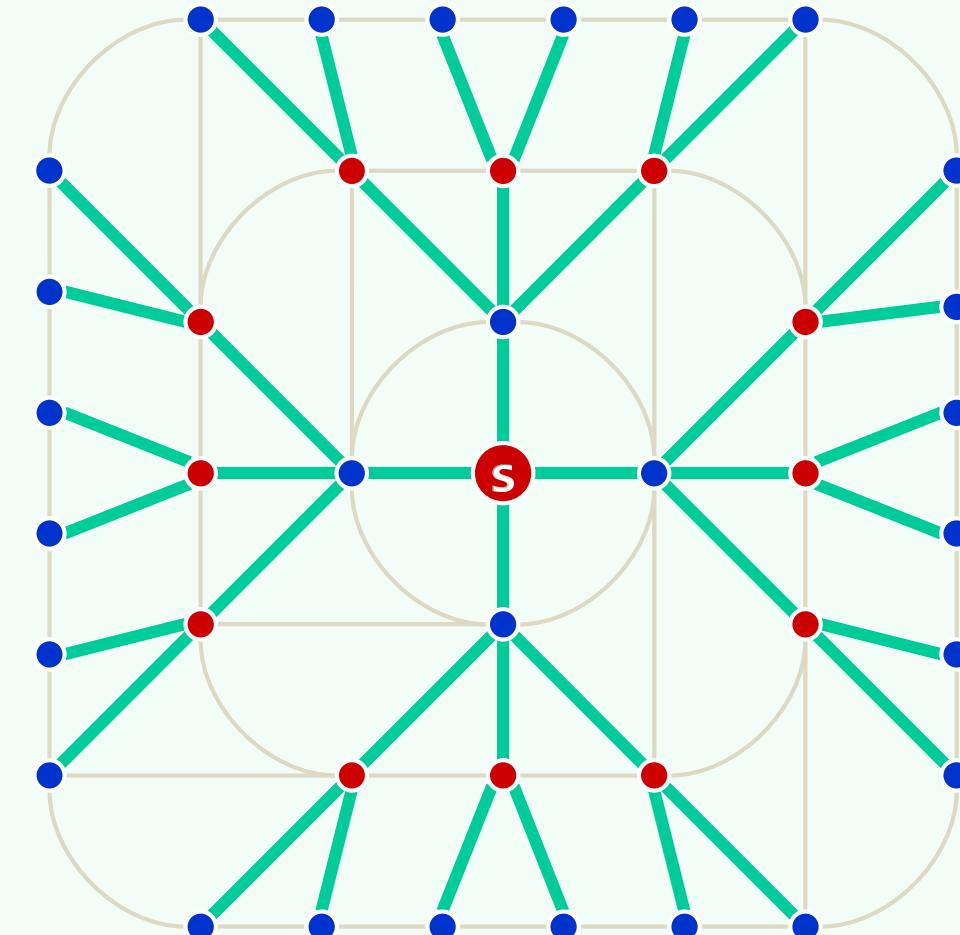
...

如此反复

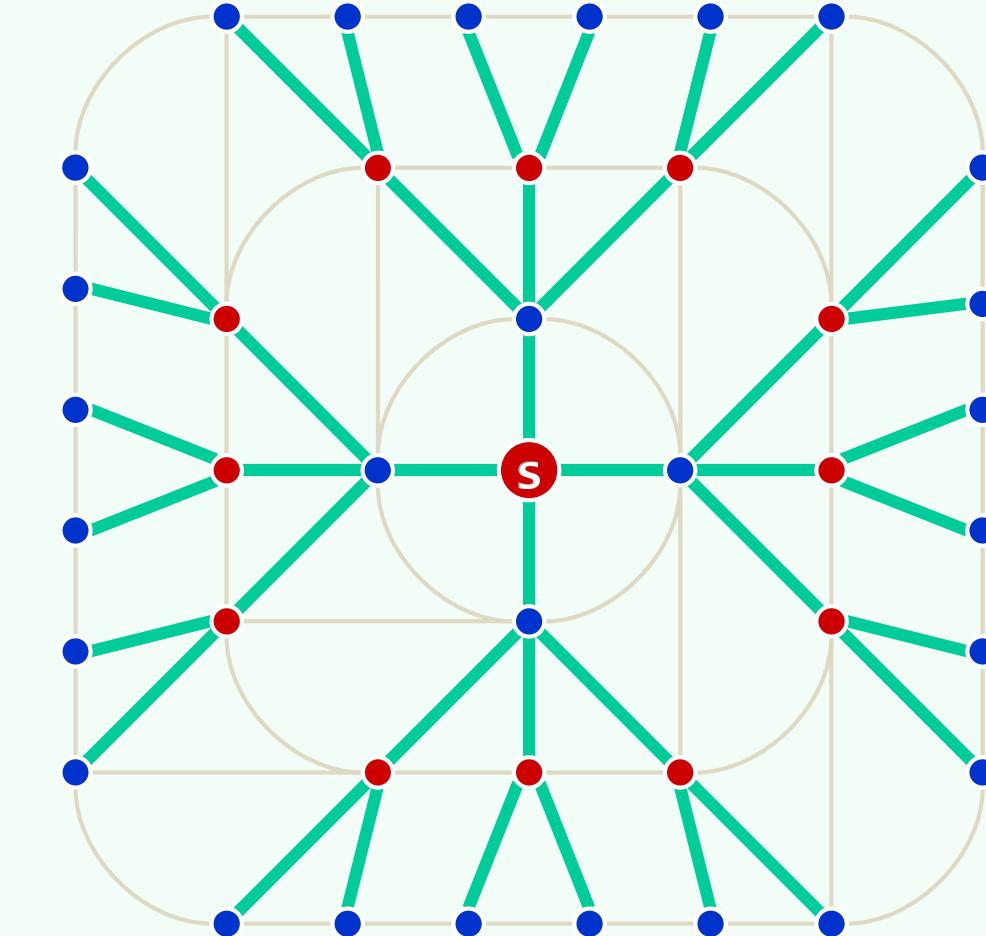
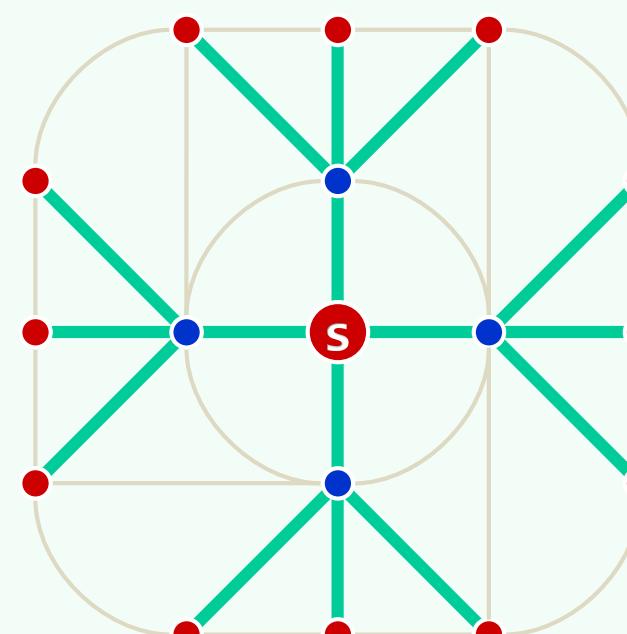
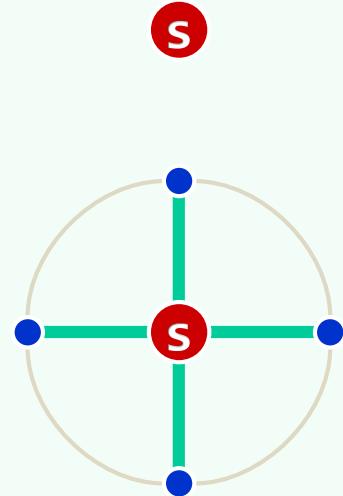
直至没有尚未访问的邻接顶点

❖ 以上策略及过程完全等同于树的层次遍历

❖ 事实上，BFS也的确会构造出原图的一棵支撑树（BFS tree）



# 譬喻：油气管线 + 绳网 + 涟漪



## Graph::BFS() [1/2]

```
template<typename Tv, typename Te> void Graph<Tv, Te>::BFS( Rank v, Rank& dClock ) {  
    Queue<Rank> Q; status(v) = DISCOVERED; Q.enqueue(v); dTime(v) = dClock++; //起点  
    for ( Rank fClock = 0; !Q.empty(); ) { //在Q变空之前, 反复地  
        if ( dTime(v) < dTime( Q.front() ) ) //dTime的增加, 意味着开启新一代, 因此  
            v //dClock++, fClock = 0; //dTime递增, fTime复位  
        v = Q.dequeue(); //取出队首顶点v, 并  
        for ( Rank u = firstNbr(v); -1 != u; u = nextNbr(v, u) ) //考查v的每一个邻居u  
            /* ... 视u的状态分别处理: 最终, 所有顶点按[dTime, fTime]字典序被遍历 ... */  
        status(v) = VISITED; fTime(v) = fClock++; //至此, v访问完毕  
    } //for  
} //BFS
```

## Graph::BFS() [2/2]

```
/* ..... */  
  
v v = Q.dequeue(); //取出队首顶点v，并  
for ( Rank u = firstNbr(v); -1 != u; u = nextNbr(v, u) ) //考查v的每一个邻居u  
u if ( UNDISCOVERED == status(u) ) { //若u尚未被发现，则发现之  
    status(u) = DISCOVERED; Q.enqueue(u); dTime(u) = dClock; //发现该顶点  
    type(v, u) = TREE; parent(u) = v; //引入树边，拓展BFS树  
}  
u u } else //若u已被发现（正在队列中），或者甚至已访问完毕（已出队列），则  
    type(v, u) = CROSS; //将(v, u)归类于跨边  
v status(v) = VISITED; fTime(v) = fClock++; //至此，v访问完毕  
/* ..... */
```

图

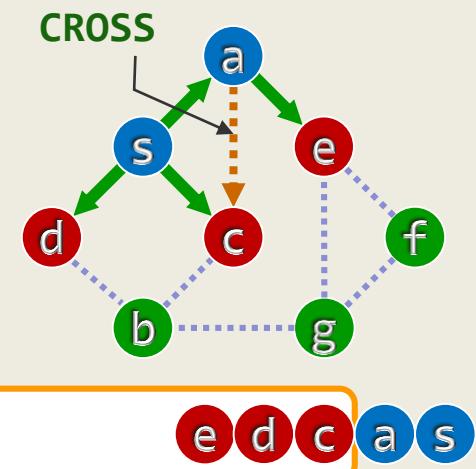
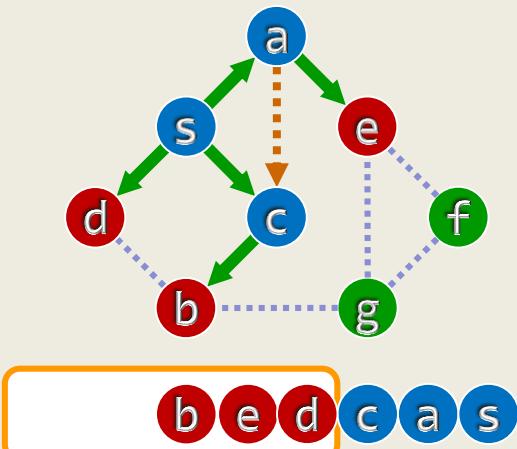
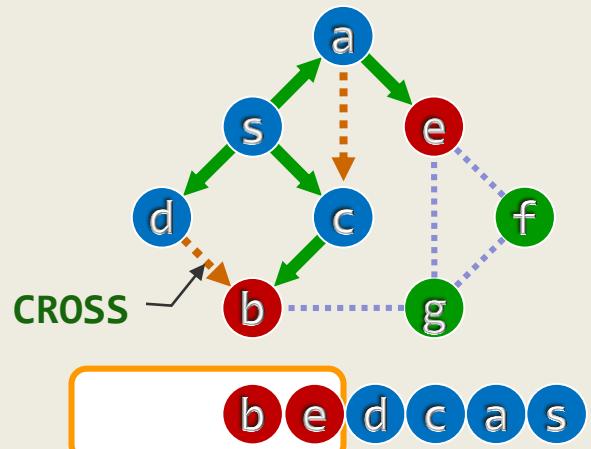
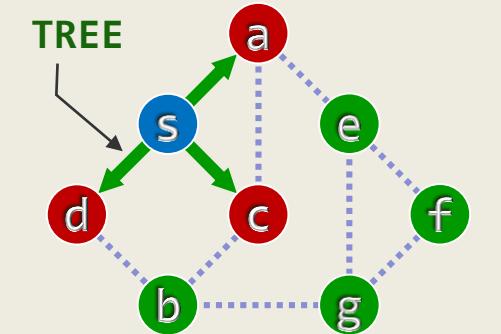
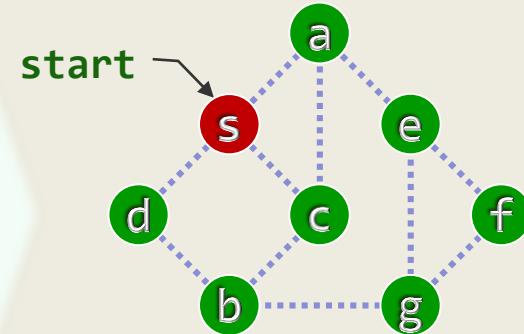
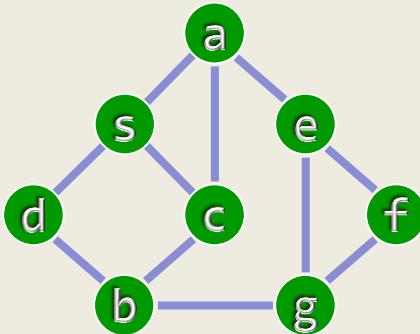
## 广度优先搜索：实例

10.-D2

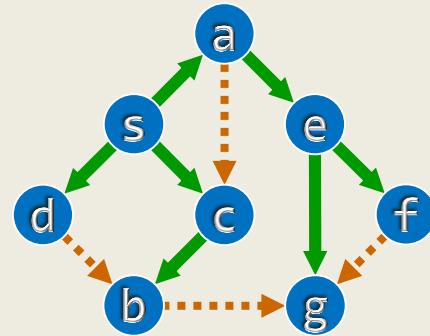
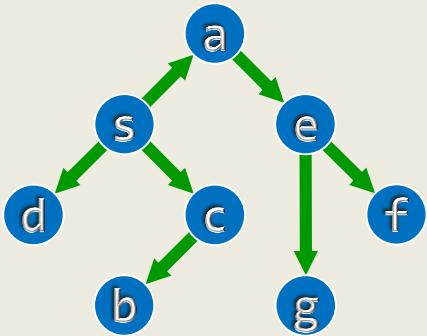
邓俊辉

deng@tsinghua.edu.cn

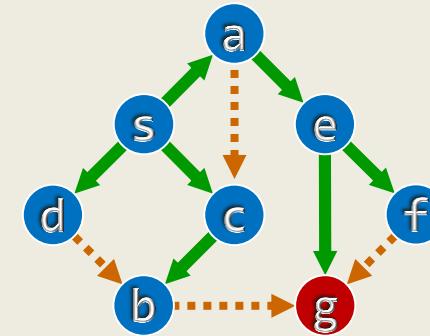
## 无向图 (1/2)



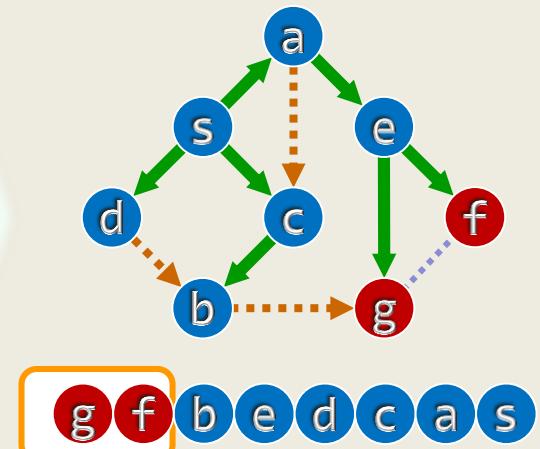
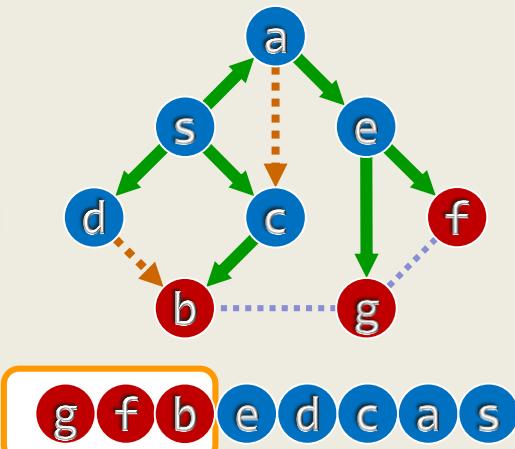
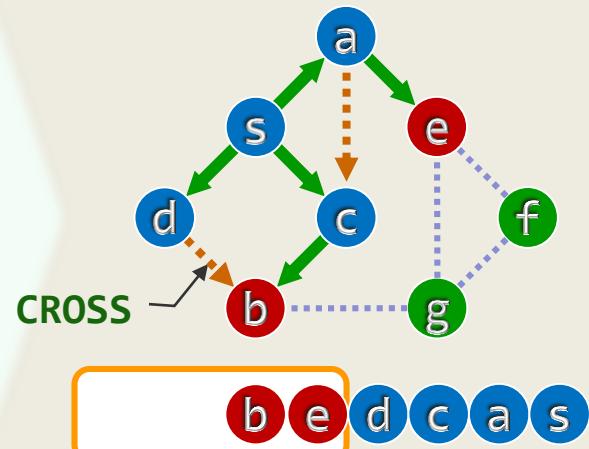
## 无向图 (2/2)



g f b e d c a s



g f b e d c a s



图

广度优先搜索：推广

10.-D3

邓俊辉

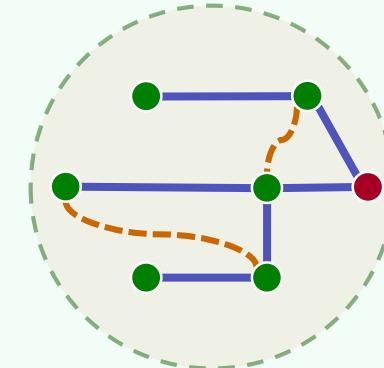
deng@tsinghua.edu.cn

一个人做一件好事并不难，难的是一辈子做好事，不做坏事

# 连通分量 + 可达分量

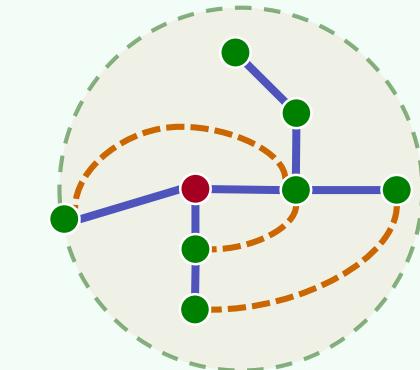
## ❖ 问题

- 给定无向图，找出其中任一顶点s所在的连通图
- 给定有向图，找出源自其中任一顶点s的可达分量

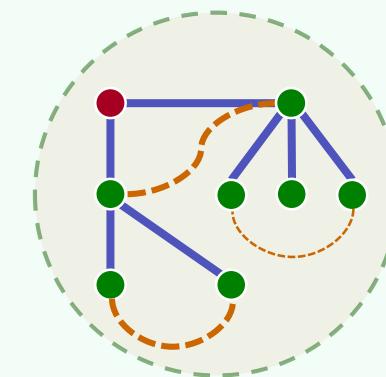


## ❖ 算法

- 从s出发做BFS
- 输出所有被发现的顶点
- 队列为空后立即终止，无需考虑其它顶点



❖ 若图中包含多个连通/可达分量，又该如何保证对全图的遍历呢？



## Graph::bfs()

```
template <typename Tv, typename Te>

void Graph<Tv, Te>::bfs( Rank s ) { //s < n

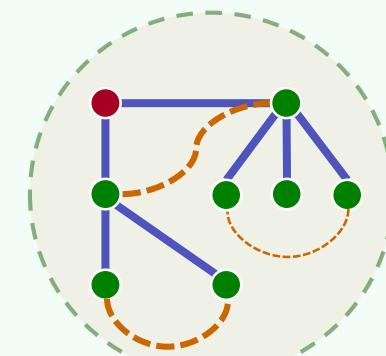
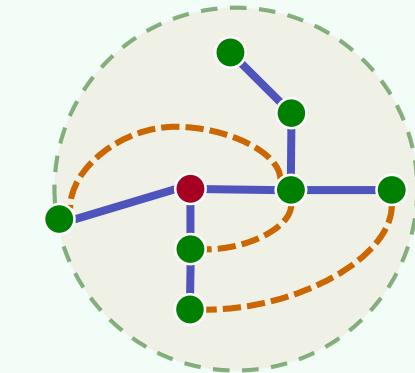
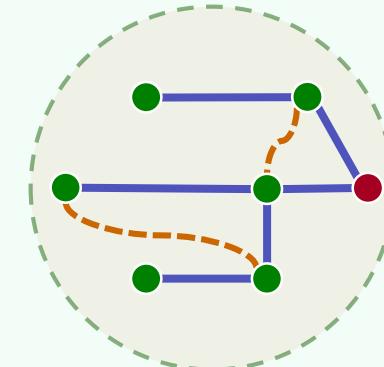
    reset(); Rank dClock = 0; //全图复位

    for ( Rank v = s; v < s + n; v++ ) //从s起顺次检查所有顶点

        if ( UNDISCOVERED == status(v % n) ) //一旦遇到尚未发现者

            BFS( v % n, dClock ); //即从它出发启动一次BFS

} //如此可完整覆盖全图，且总体复杂度依然保持为 $\Theta(n+e)$ 
```



# 复杂度

❖ 考查无向图...

❖ `bfs()`的初始化 (reset()) :  $\mathcal{O}(n + e)$

❖ `BFS()`的迭代

- 外循环 (`while ( !Q.empty() )`)

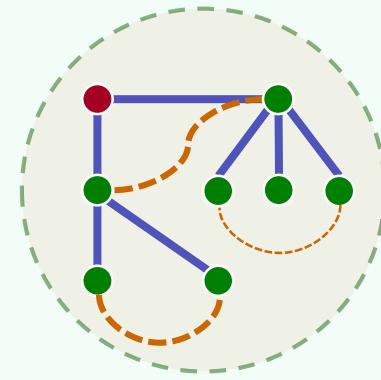
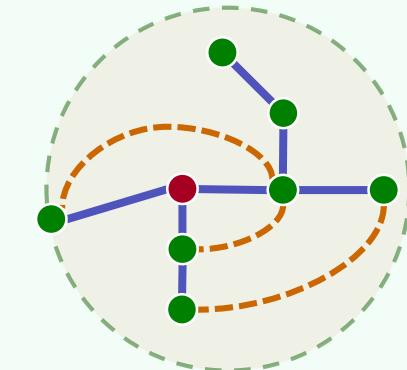
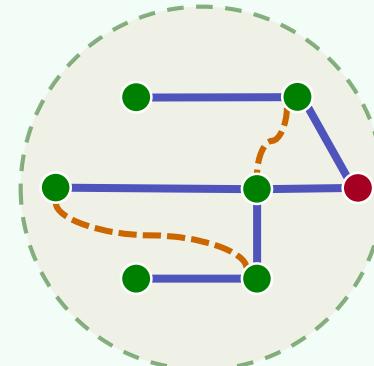
每个顶点各进入1次

- 内循环 (枚举v的每一邻居) :  $\mathcal{O}(1 + \deg(v))$  (改用邻接表)

- 总共:  $\mathcal{O}(\sum_{v \in V} (1 + \deg(v))) = \mathcal{O}(n + 2e)$

❖ 整个算法:  $\mathcal{O}(n + e) + \mathcal{O}(n + 2e) = \mathcal{O}(n + e)$

❖ 有向图呢? 亦是如此!



图

## 广度优先搜索：性质及规律

10-D4

邓俊辉

deng@tsinghua.edu.cn

闭门求学，其学无用。欲从天下国家万事万物而学之，则汗漫九垓，遍游四宇尚已

## 边分类

- 经BFS后，所有边将确定方向，且被分为两类
- (v, u)被标记为TREE时，v为DISCOVERED且u为UNDISCOVERED



- (v, u)被标记为CROSS时，v和u均为DISCOVERED 或者 v为DISCOVERED而u为VISITED



不论(v, u)是有向边或无向边，两种情况均可能出现

# BFS树/森林

❖ 对于（起始于 $v$ 的）每一连通/可达分量， $bfs()$ 进入 $BFS(v)$ 恰好1次

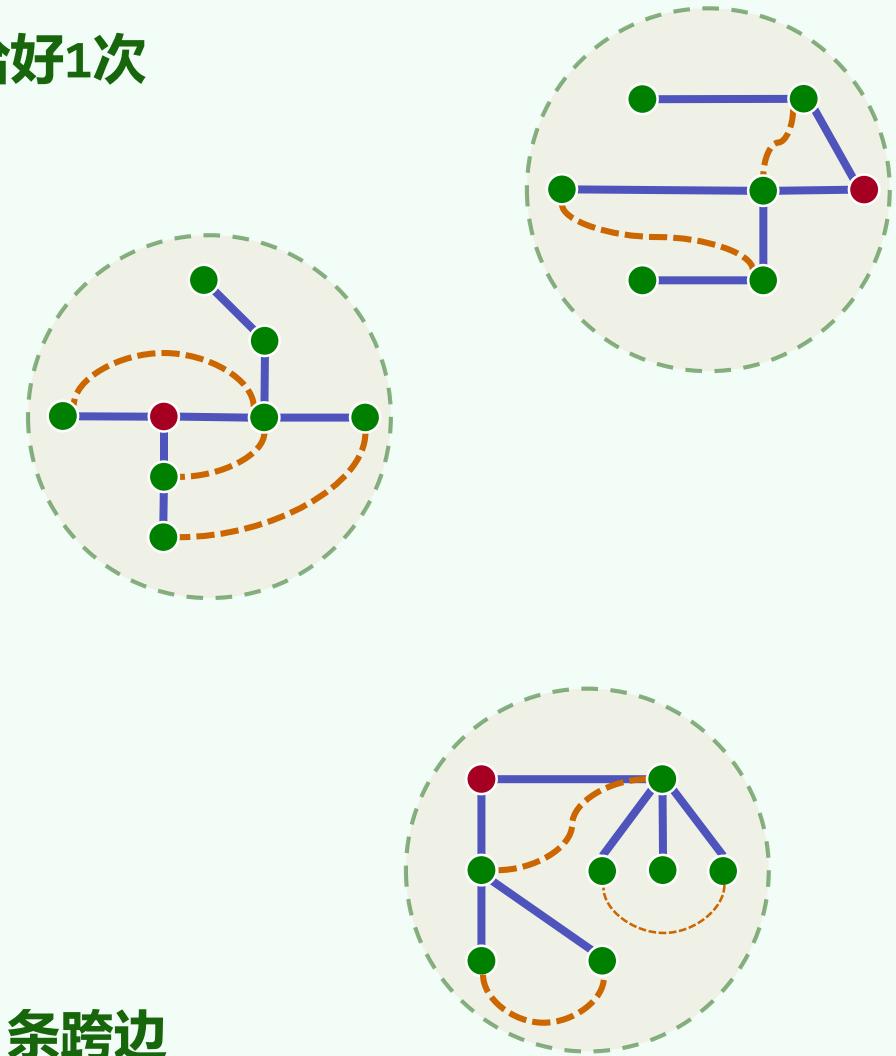
❖ 进入 $BFS(v)$ 时，队列为空； $v$ 所属分量内的每个顶点

- 迟早会以UNDISCOVERED状态进队1次
- 进队后随即转为DISCOVERED状态，并生成一条树边
- 迟早会出队并转为VISITED状态

退出 $BFS(v)$ 时，队列为空

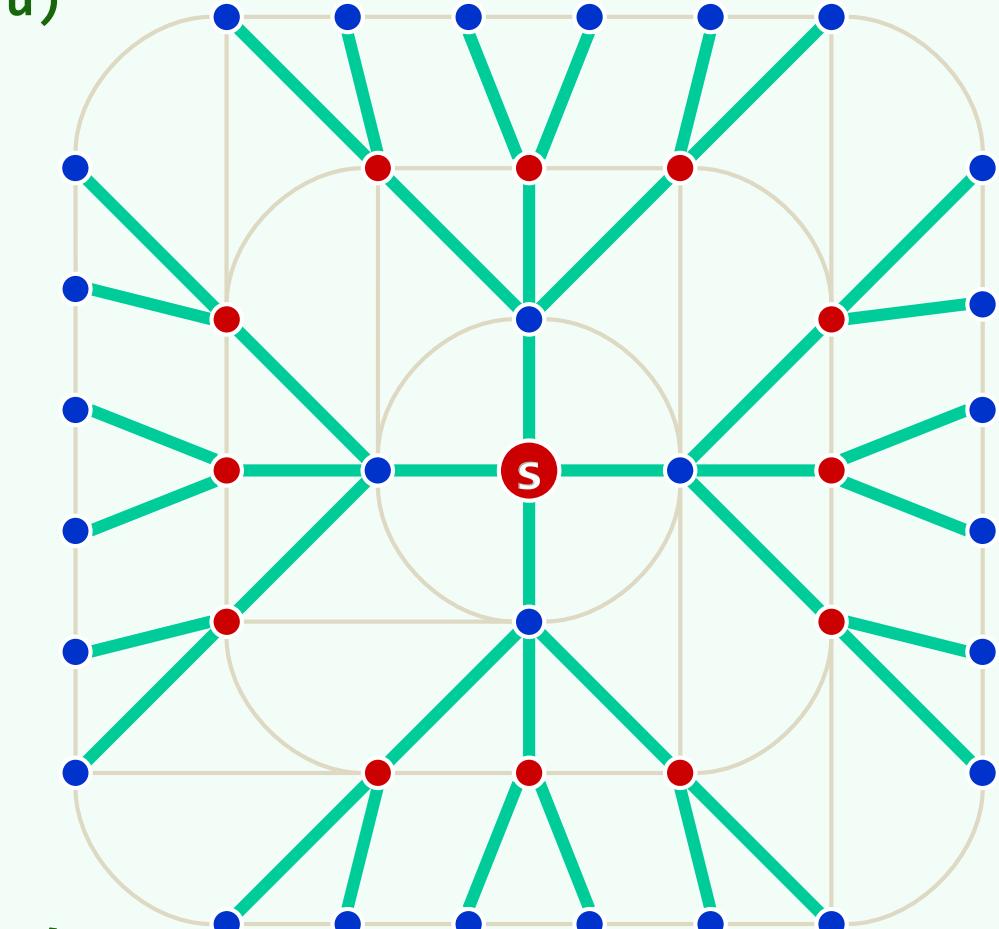
❖  $BFS(v)$ 以 $v$ 为根，生成一棵BFS树

❖  $bfs()$ 生成一个BFS森林包含  $c$  棵树、 $n-c$  条树边和  $e-n+c$  条跨边



# 最短路径

- ❖ 在无向图中，顶点 $v$ 到 $u$ 的（最近）距离记作 $\text{dist}(v, u)$
- ❖ BFS过程中，队列 $Q$ 犹如一条贪吃蛇
  - 其中的顶点按 $\text{dist}(s)$ 单调排列
  - 相邻顶点的 $\text{dist}(s)$ 相差不超过1
  - 首、末顶点的 $\text{dist}(s)$ 相差不超过1
  - 由树边联接的顶点， $\text{dist}(s)$ 恰好相差1
  - 由跨边联接的顶点， $\text{dist}(s)$ 至多相差1
- ❖ BFS树中从 $s$ 到 $v$ 的路径，即是二者在原图中的最短通路



图

## 广度优先搜索：应用

10.-D5

邓俊辉

deng@tsinghua.edu.cn

啊，五环，你比四环多一环；啊，五环，你比六环少一环

## Erdős Number

❖ Describes the "collaborative distance"

between mathematician Paul Erdős and another person,  
as measured by authorship of mathematical papers



# Chow Number

❖ chow (吴孟达) = 1

[整蛊专家] (1) \*周星驰 +吴孟达 成奎安 刘德华 关之琳 邱淑贞

❖ chow (葛优) = 2

[没完没了] (2) +葛优 \*吴倩莲 傅彪

[97家有喜事] (1) \*周星驰 +吴倩莲 吴镇宇 钟丽缇 伍咏薇 黄百鸣

❖ chow (姜昆) = 3

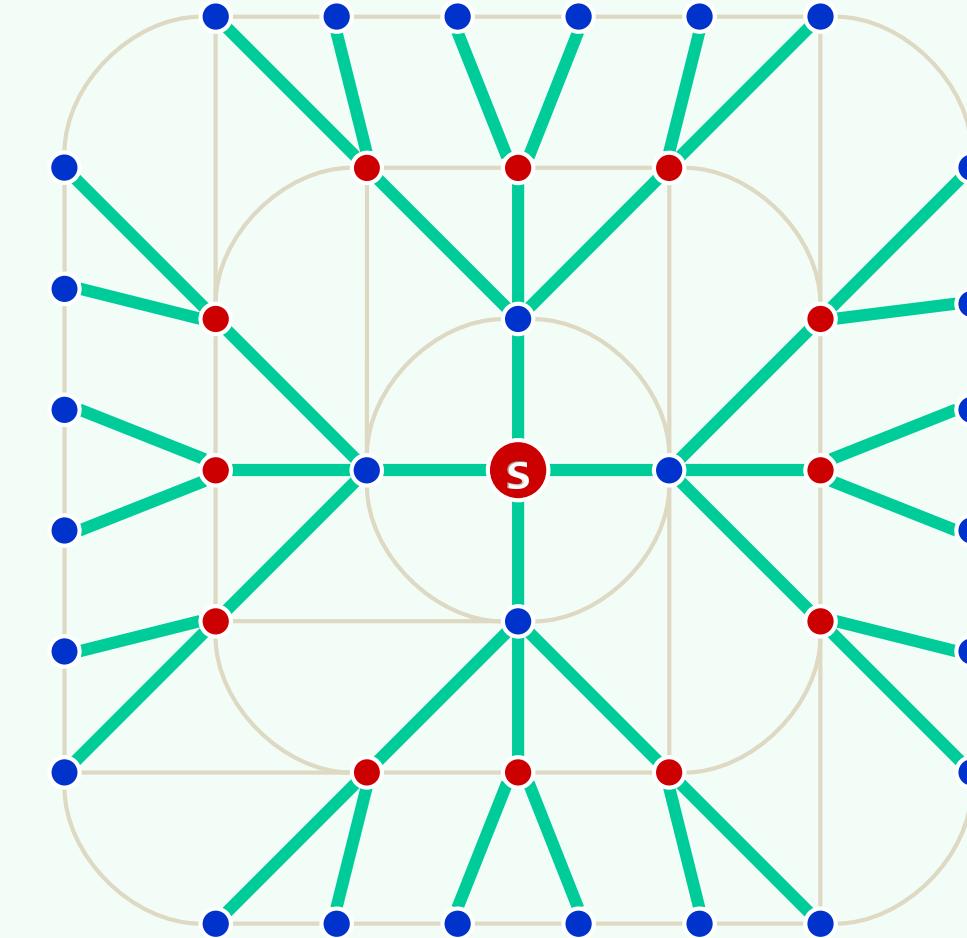
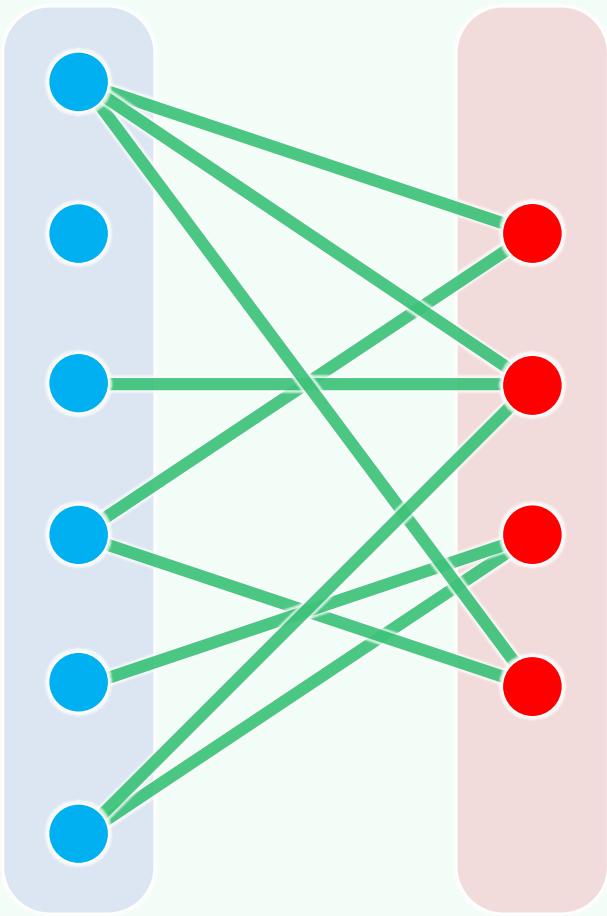
[京都球侠] (3) \*张丰毅 孙敏 +姜昆 陈佩斯 于绍康 唐杰忠

[热线追击] (2) +张丰毅 任达华 王馨平 \*吴家丽

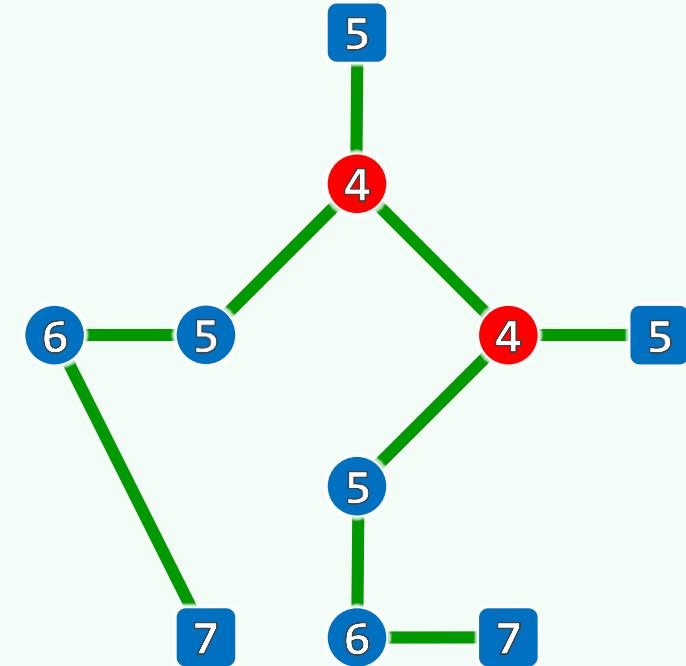
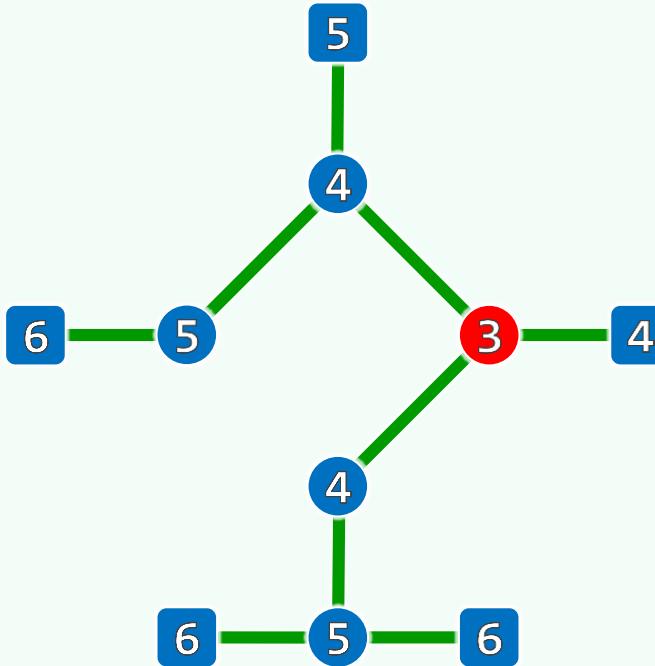
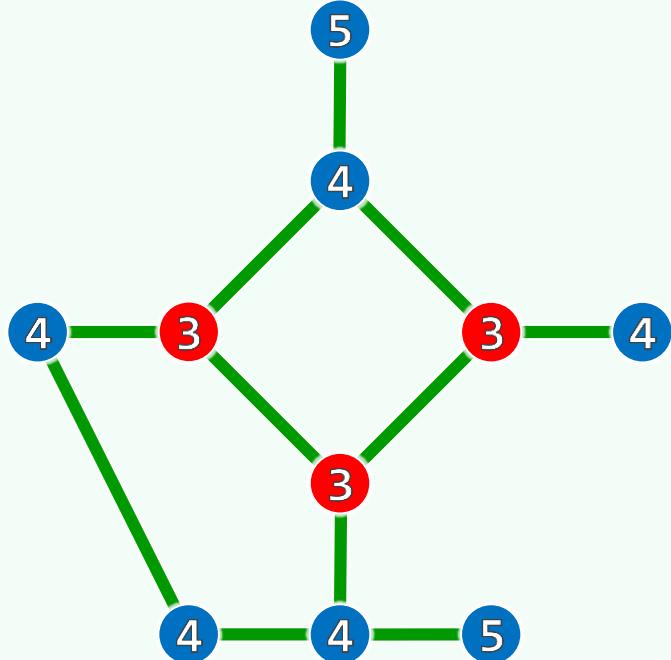
[审死官] (1) \*周星驰 吴孟达 +吴家丽 秦沛 朱咪咪 梅艳芳

❖ chow ("Julia Roberts") = Infinity

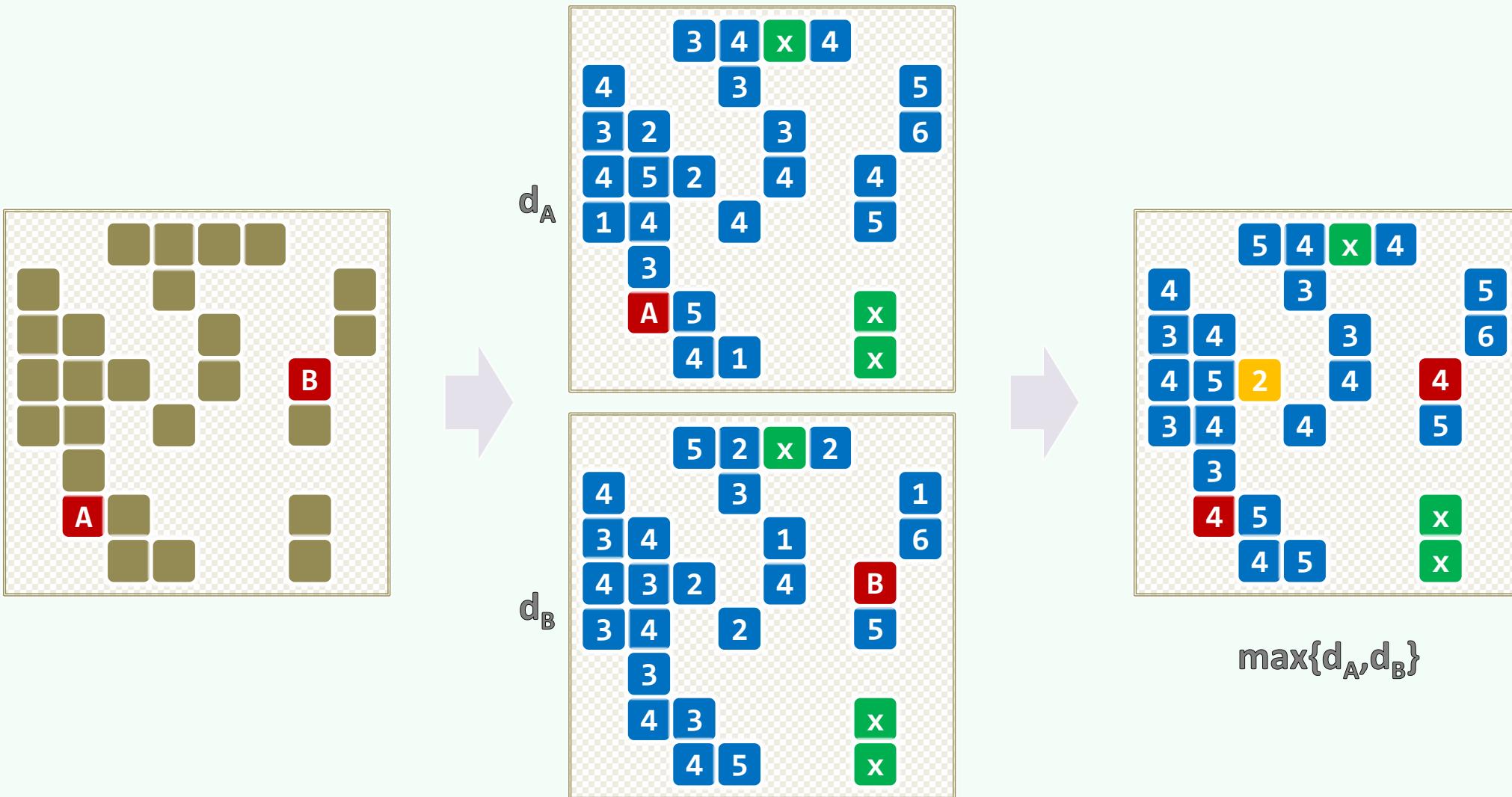
# Bipartite Graph (Bigraph)



## Eccentricity/Radius/Diameter/Center



# Knights of the Round Table



图

## 深度优先搜索：算法

10-E1

E1

Keep it simple, stupid. - K. Johnson

如右边天气阴沉多雨，我便往左边走；如遇上不宜骑马的地方，  
我就停下。如此这般行路.....我身后是否还有什么东西可看？  
有，我便返回去，因为那也是我要走的路

邓俊辉

deng@tsinghua.edu.cn

# Depth-First Search

❖  $\text{DFS}(s)$  //始自顶点s的深度优先搜索

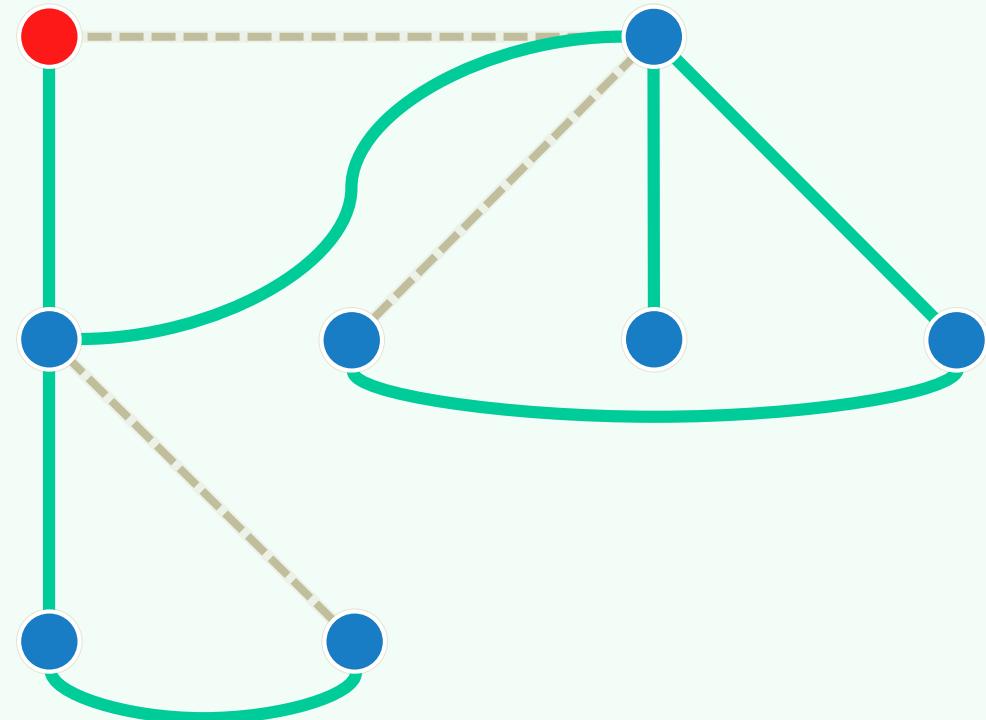
- 访问顶点s
- 若s尚有未被访问的邻居，则任取其一u，递归执行 $\text{DFS}(u)$
- 否则，返回

❖ 若此时尚有顶点未被访问

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

❖ 对树而言，等效于先序遍历：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> void Graph<Tv, Te>::DFS( Rank v, Rank& clock ) {  
    dTime(v) = ++clock; status(v) = DISCOVERED; //发现当前顶点v  
    v for ( Rank u = firstNbr(v); -1 != u; u = nextNbr(v, u) ) //考察v的每一邻居u  
        switch ( status(u) ) { //并视其状态分别处理  
            u case UNDISCOVERED: //u尚未发现, 意味着支撑树可在此拓展  
                type(v, u) = TREE; parent(u) = v; DFS( u, clock ); break; //递归  
            u case DISCOVERED: //u已被发现但尚未访问完毕, 应属被后代指向的祖先  
                type(v, u) = BACKWARD; break;  
            u 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  
                type(v, u) = dTime(v) < dTime(u) ? FORWARD : CROSS; break;  
        } //switch  
    v status(v) = VISITED; fTime(v) = ++clock; //至此, 当前顶点v方告访问完毕  
}
```

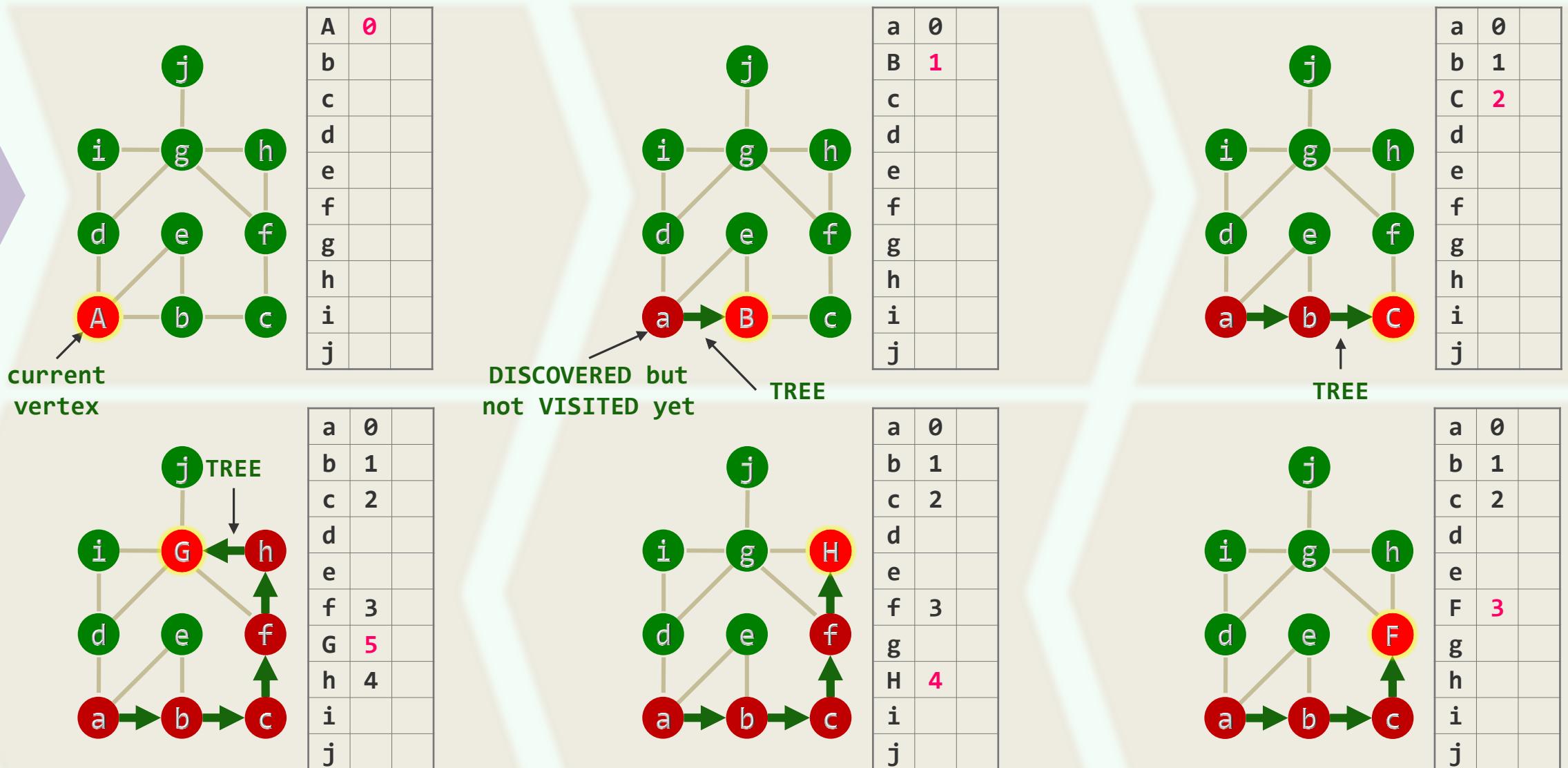
图

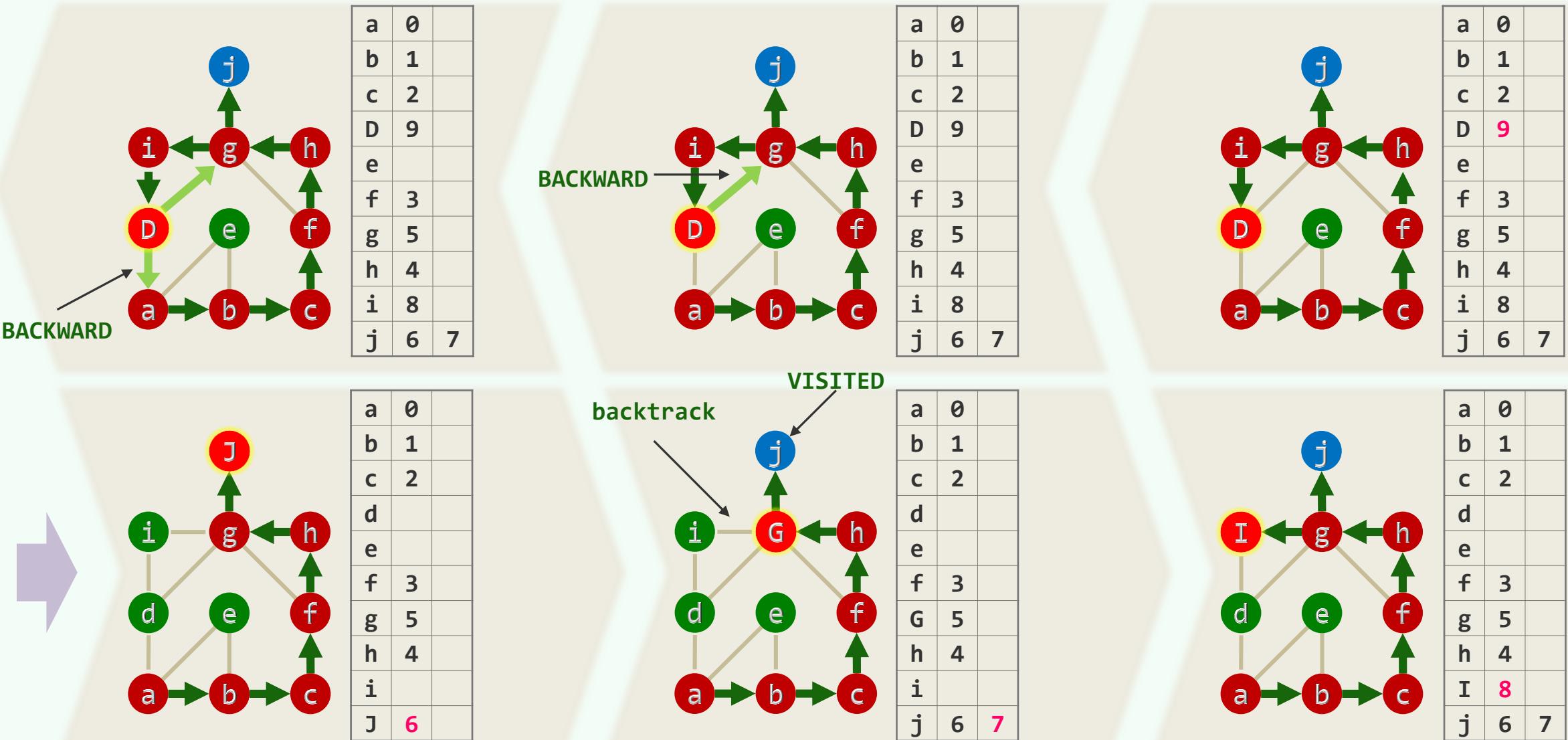
## 深度优先搜索：实例（无向图）

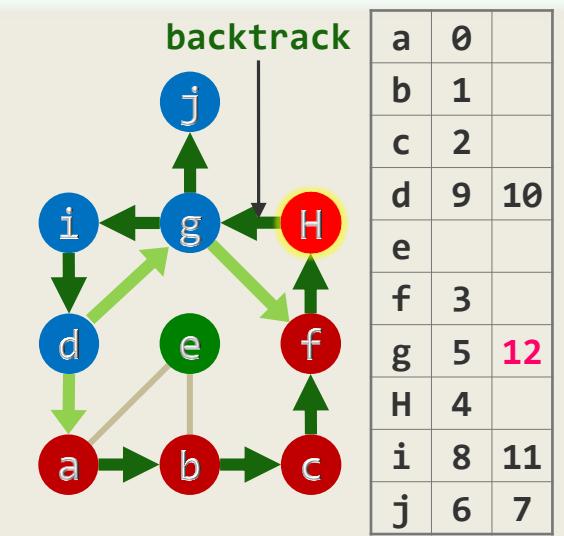
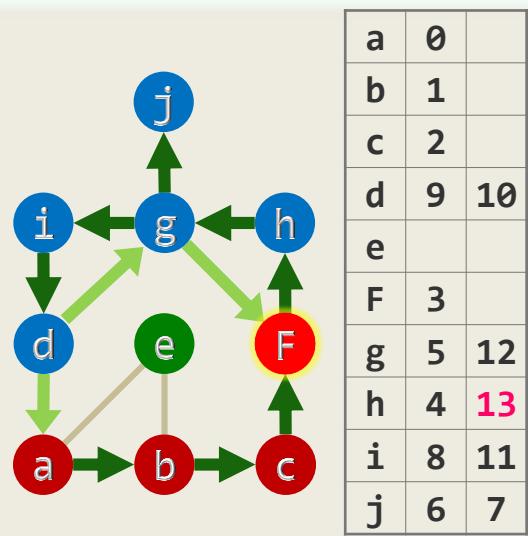
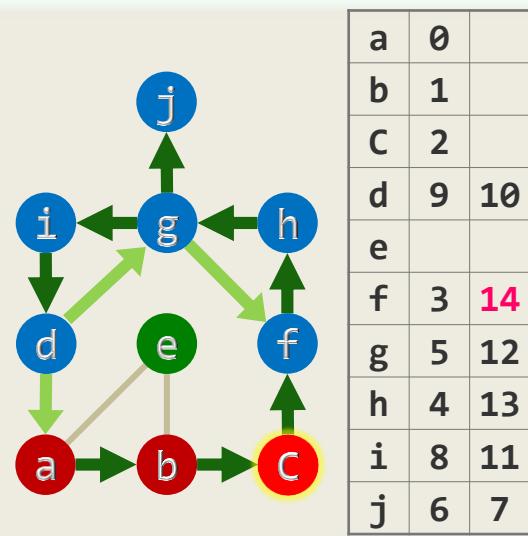
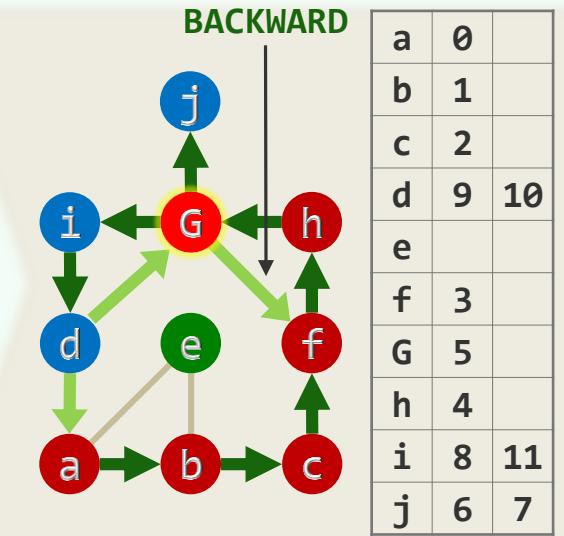
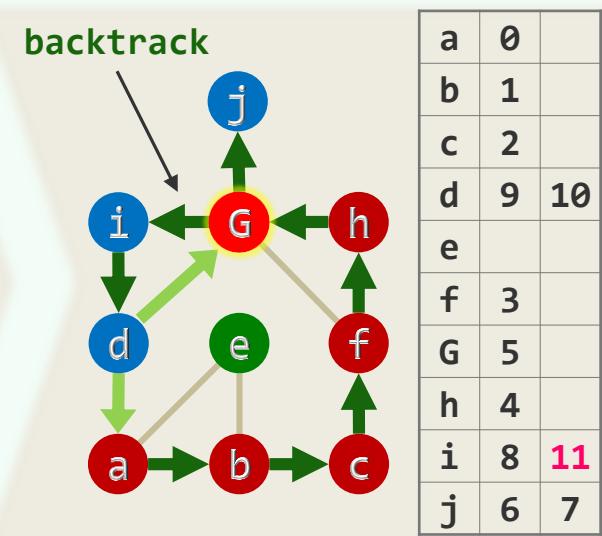
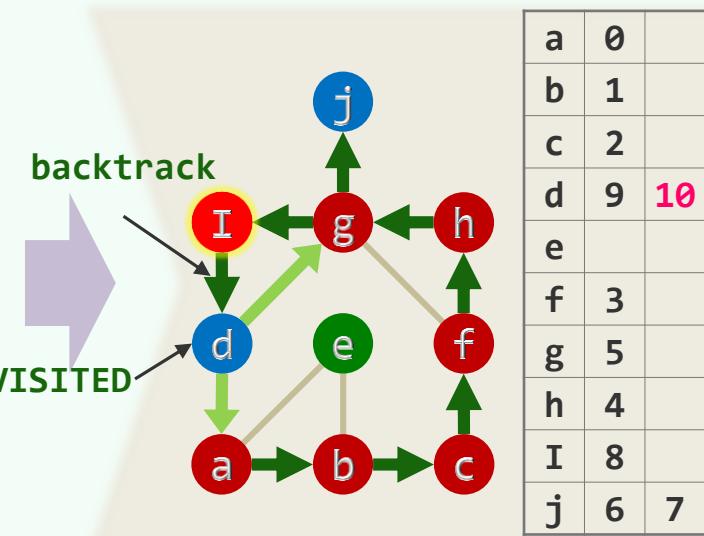
10-E2

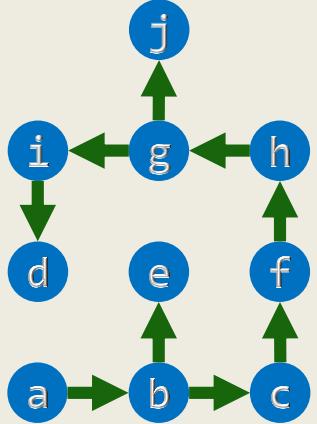
悔相道之不察兮，延伫乎吾将反  
回朕车以复路兮，及行迷之未远

邓俊辉  
[deng@tsinghua.edu.cn](mailto:deng@tsinghua.edu.cn)

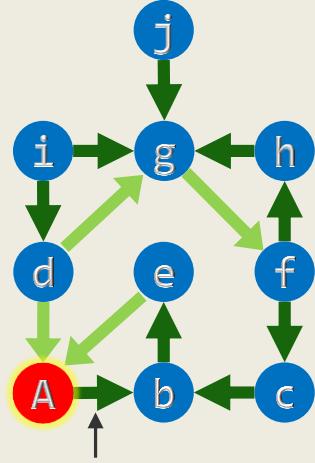




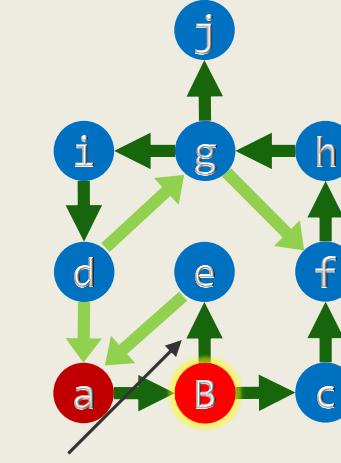




a	0	19
b	1	18
c	2	15
d	9	10
e	16	17
f	3	14
g	5	12
h	4	13
i	8	11
j	6	7

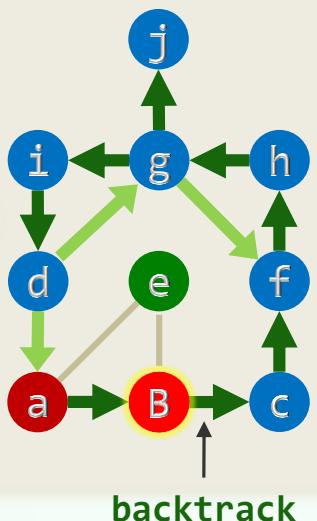


A	0	
b	1	18
c	2	15
d	9	10
e	16	17
f	3	14
g	5	12
h	4	13
i	8	11
j	6	7



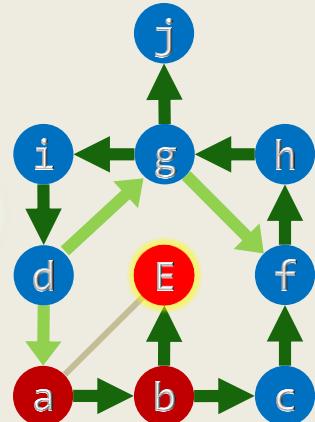
backtrack

a	0	
B	1	
c	2	15
d	9	10
e	16	17
f	3	14
g	5	12
h	4	13
i	8	11
j	6	7

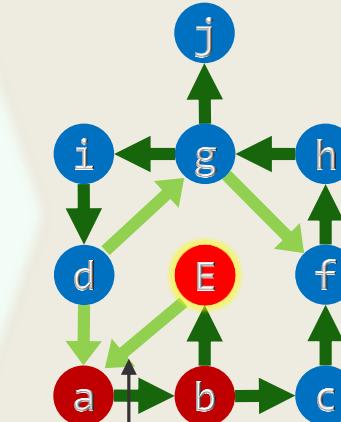


backtrack

a	0	
B	1	
c	2	15
d	9	10
e		
f	3	14
g	5	12
h	4	13
i	8	11
j	6	7



a	0	
b	1	
c	2	15
d	9	10
E	16	
f	3	14
g	5	12
h	4	13
i	8	11
j	6	7



BACKWARD

a	0	
b	1	
c	2	15
d	9	10
E	16	
f	3	14
g	5	12
h	4	13
i	8	11
j	6	7

图

深度优先搜索：推广

10-E3

邓俊辉

deng@tsinghua.edu.cn

我要到N进K学堂去了，仿佛是想走异路，逃异地，去寻求别样的人们

# 非连通

❖ 与BFS( $v$ )类似，DFS( $v$ )也可遍历 $v$ 所属分量

——若含多个分量呢？

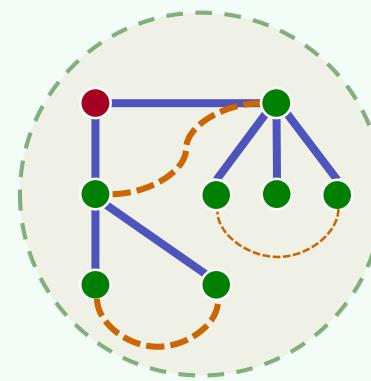
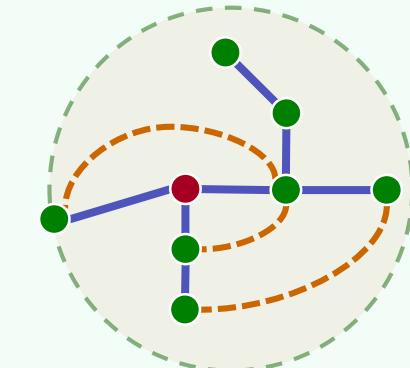
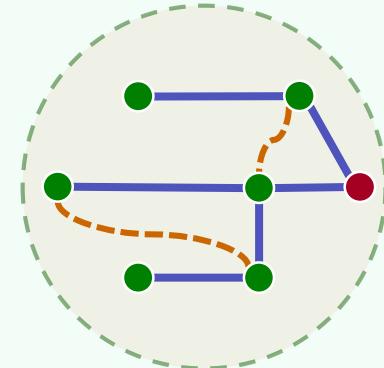
❖ 与bfs( $s$ )类似（采用邻接表）

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）



## Graph::dfs()

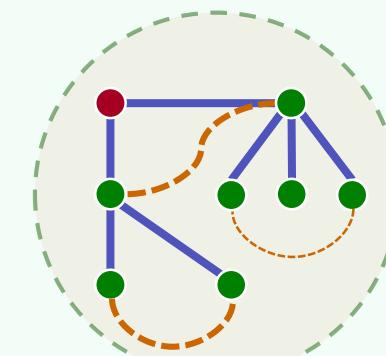
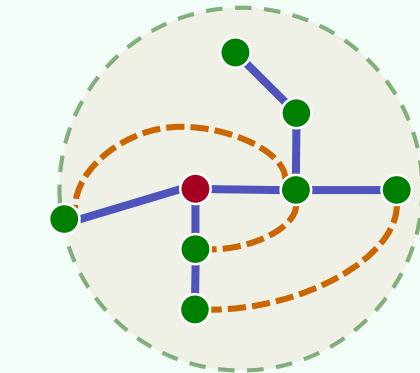
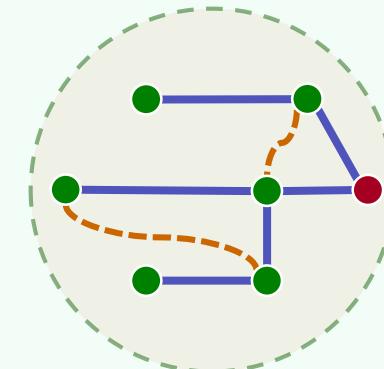
```
template <typename Tv, typename Te>

void Graph<Tv, Te>::dfs( Rank s ) { //s < n
    reset(); Rank clock = 0; //全图复位

    for ( Rank v = s; v < s + n; v++ ) //从s起顺次检查所有顶点

        if ( UNDISCOVERED == status(v % n) ) //一旦遇到尚未发现者
            DFS( v % n, clock ); //即从它出发启动一次DFS

} //如此可完整覆盖全图，且总体复杂度依然保持为 $\Theta(n+e)$ 
```



图

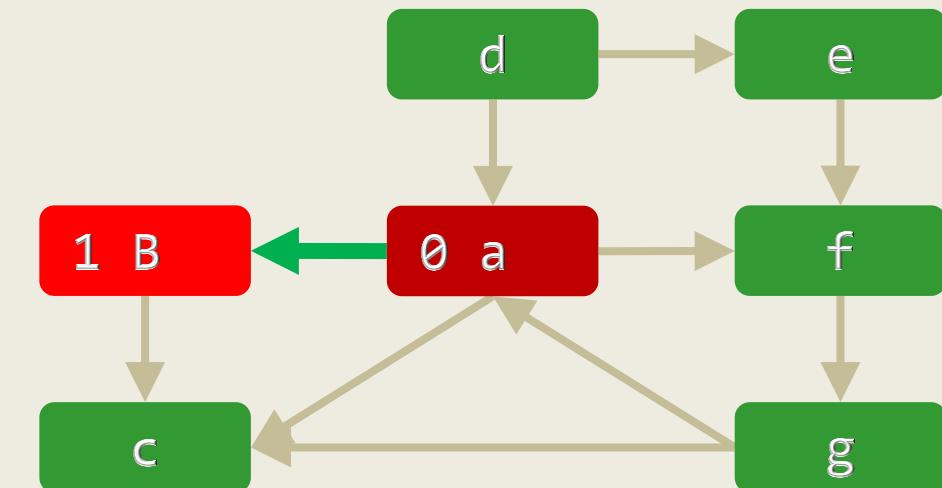
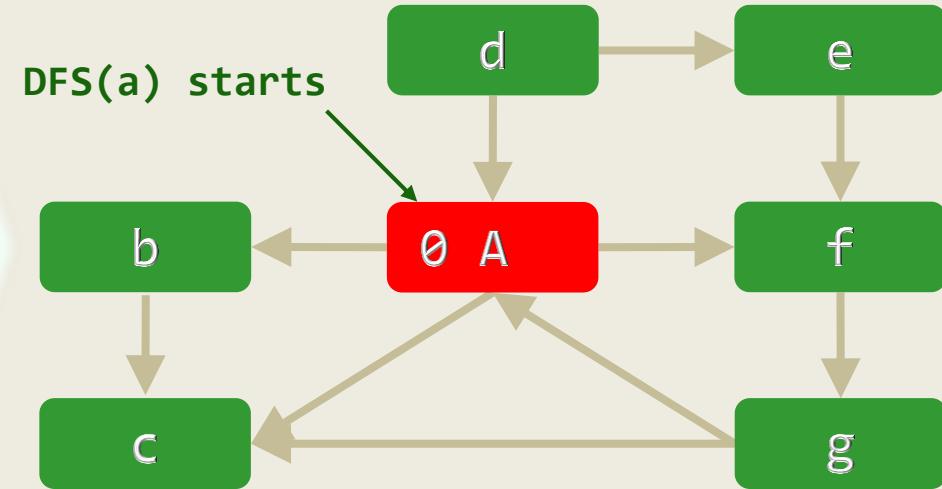
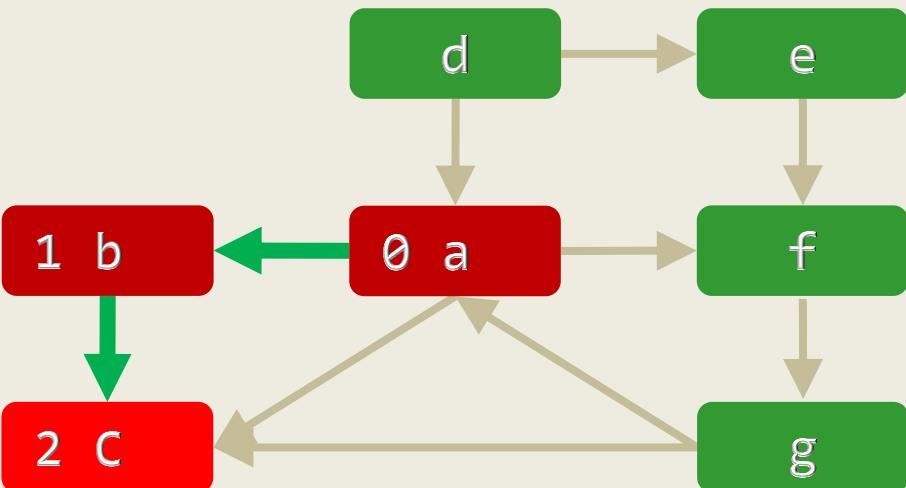
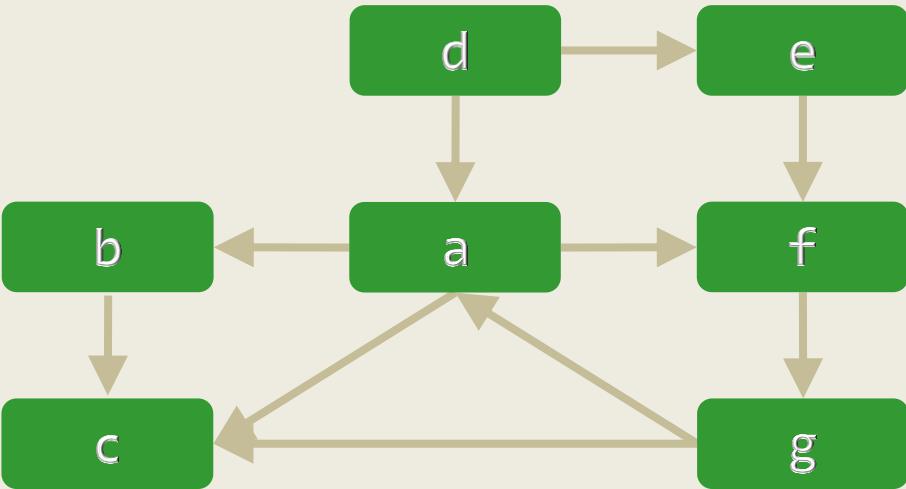
## 深度优先搜索：实例（有向图）

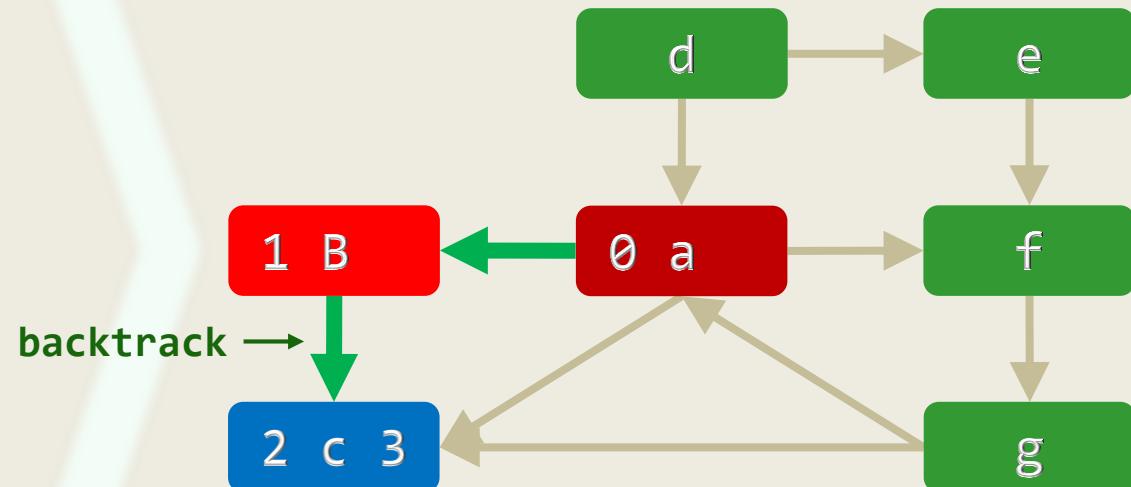
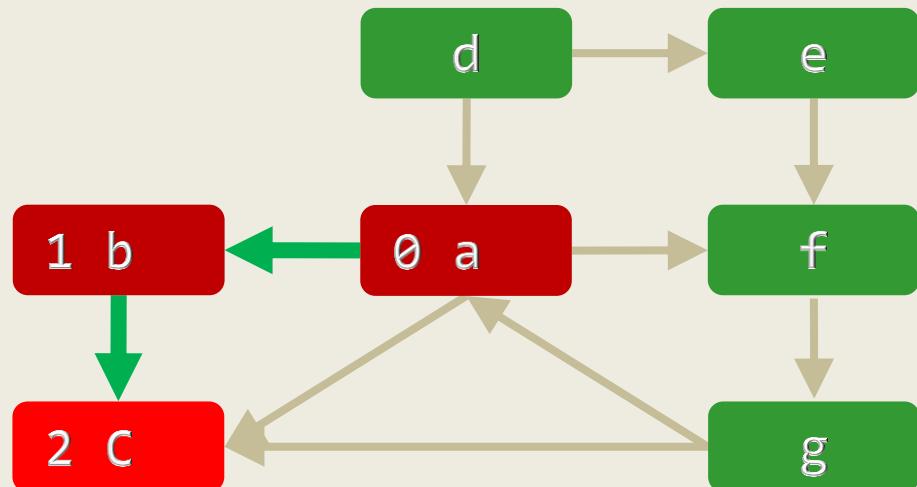
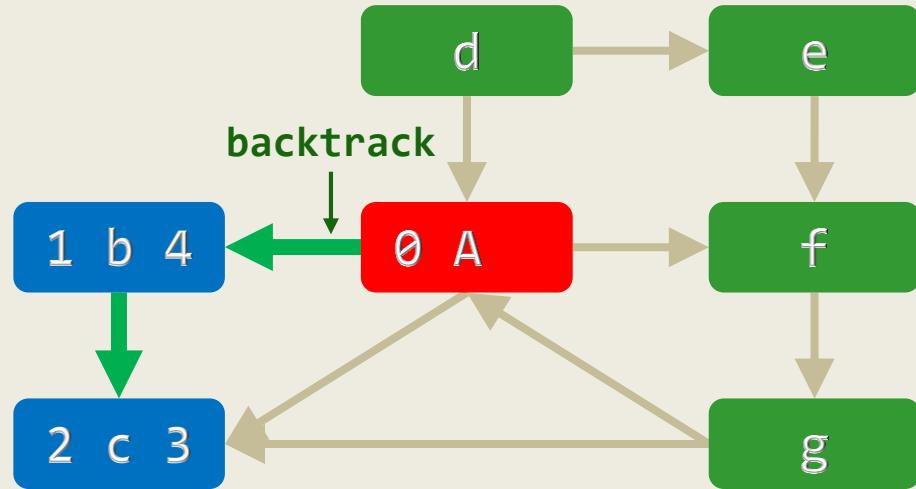
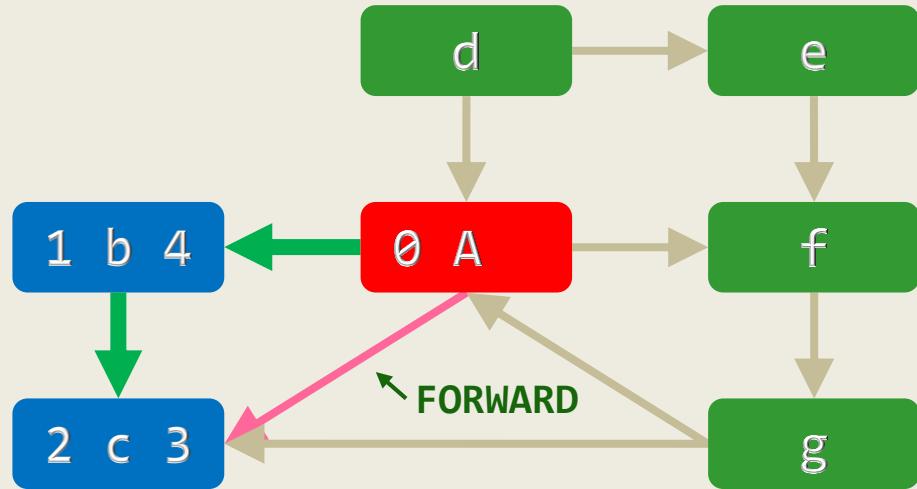
10-E4

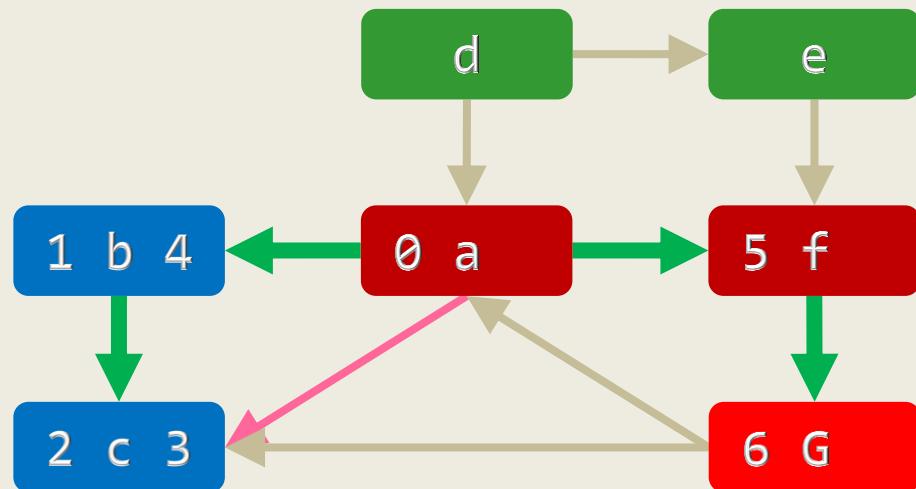
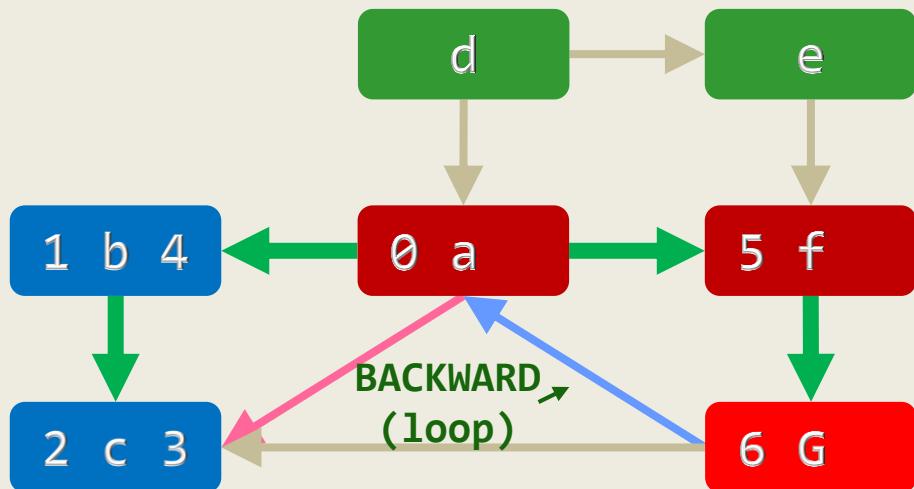
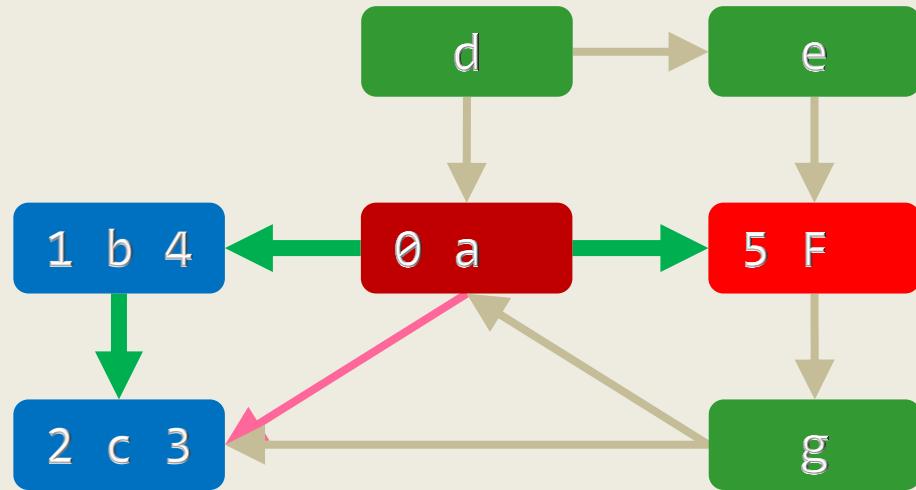
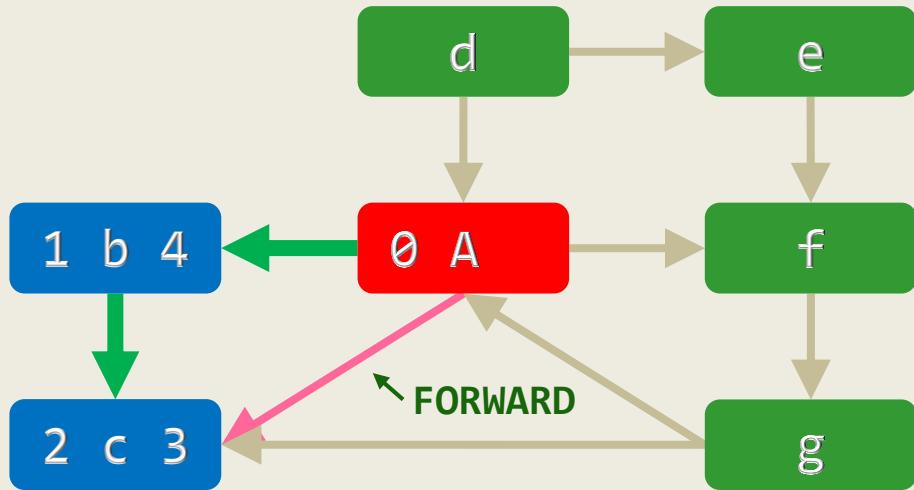
邓俊辉

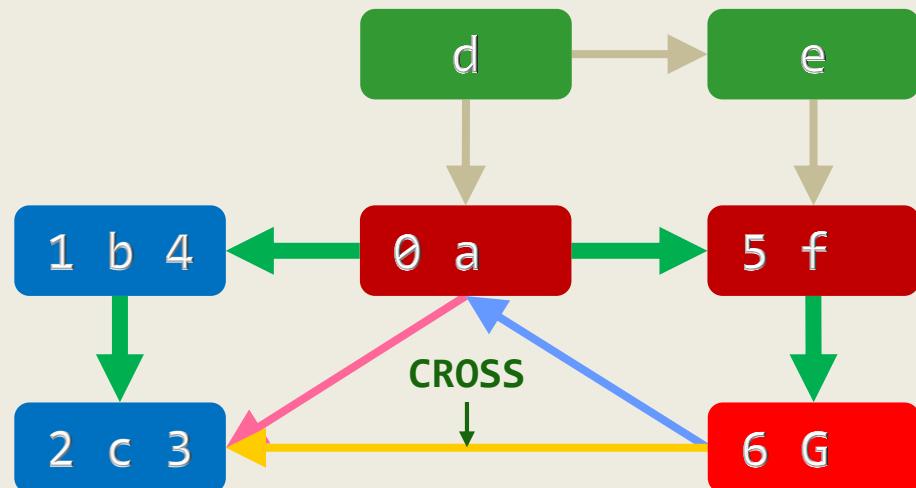
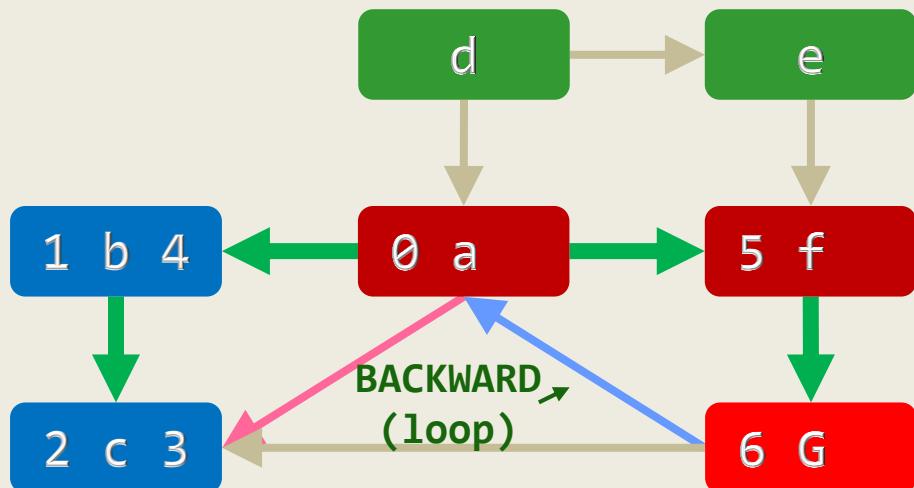
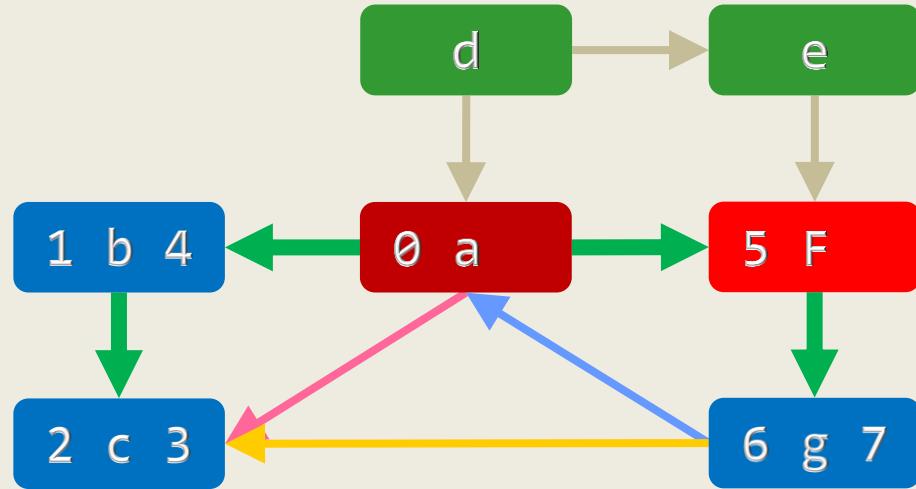
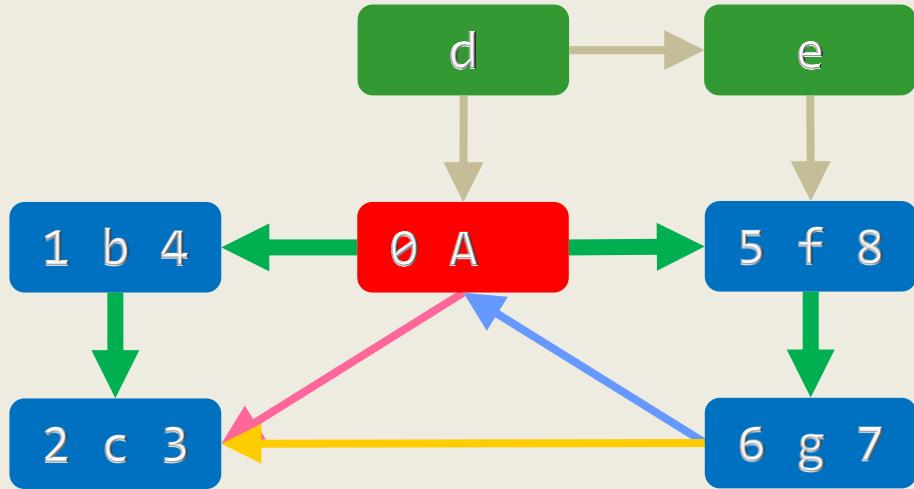
deng@tsinghua.edu.cn

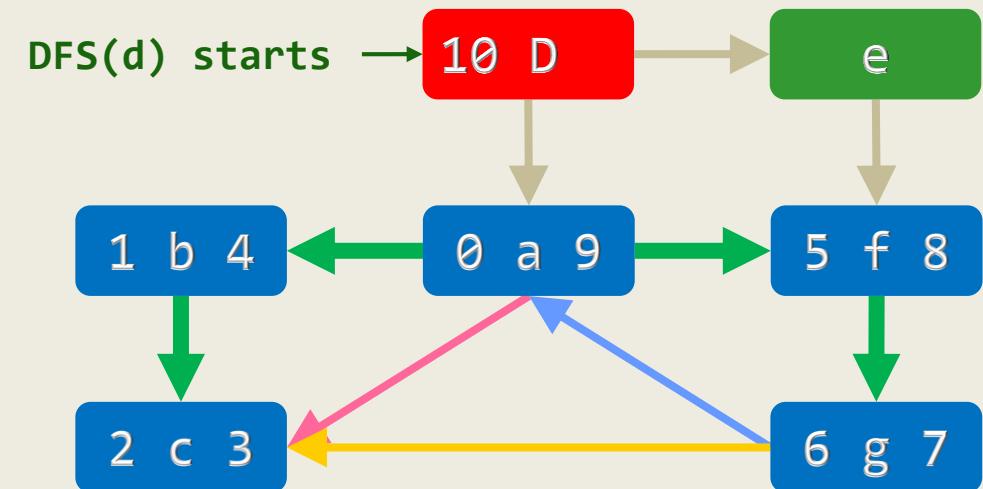
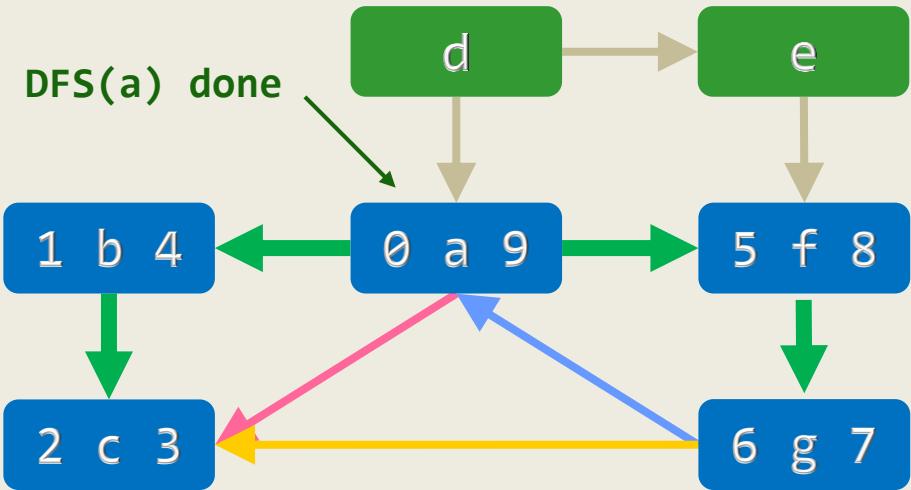
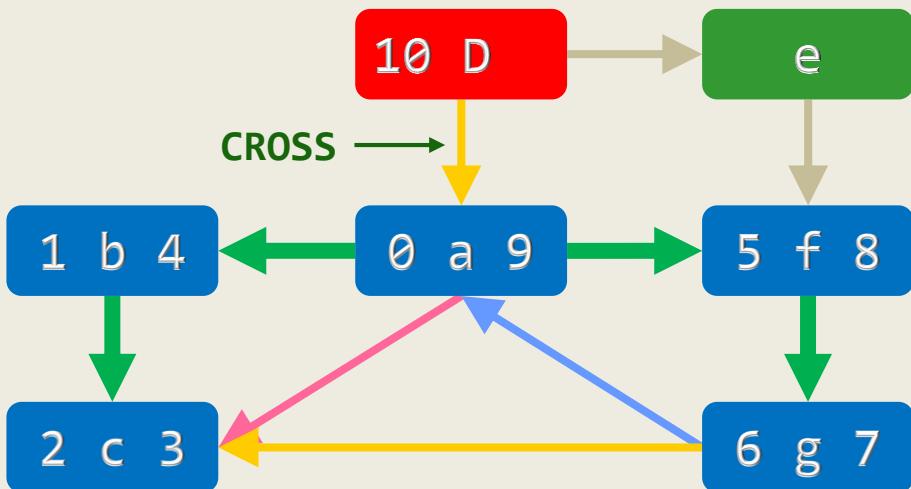
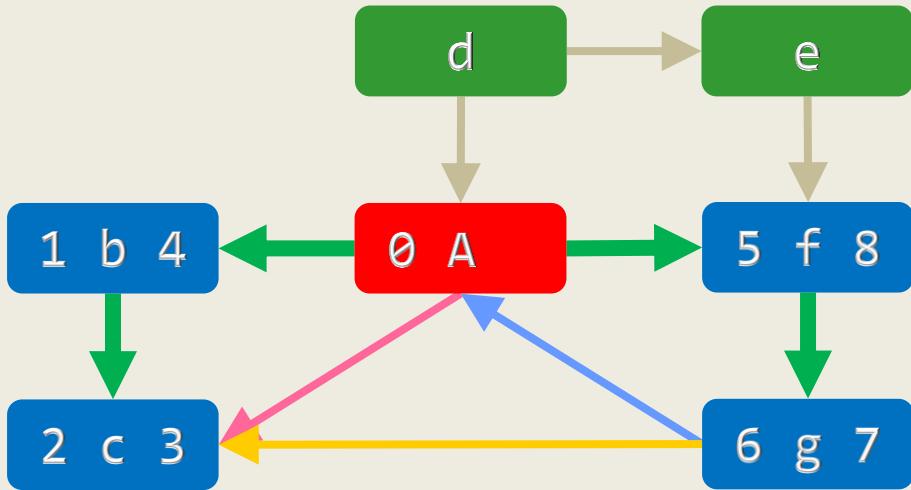
時率意獨駕，不由徑路；車跡所窮，輒慟哭而反

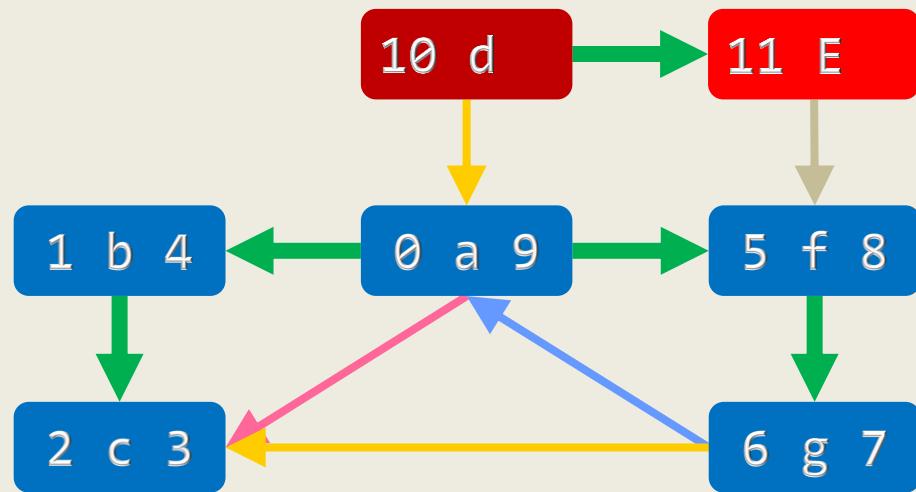
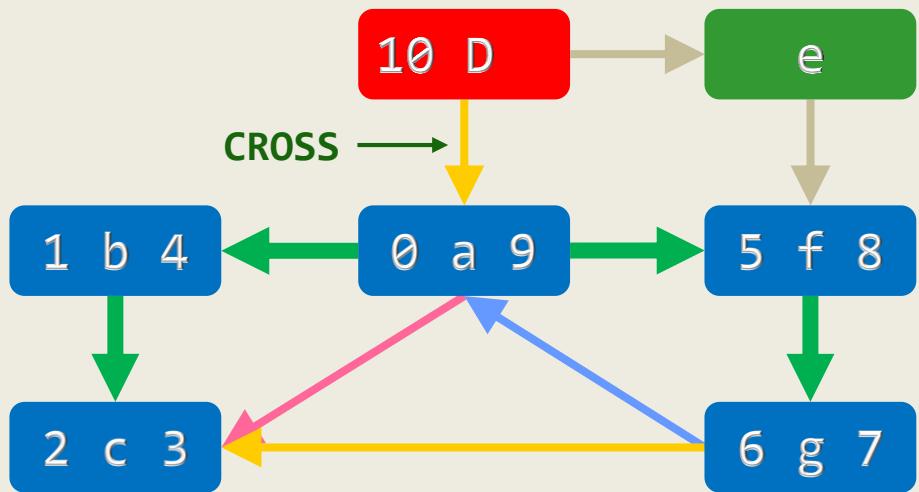
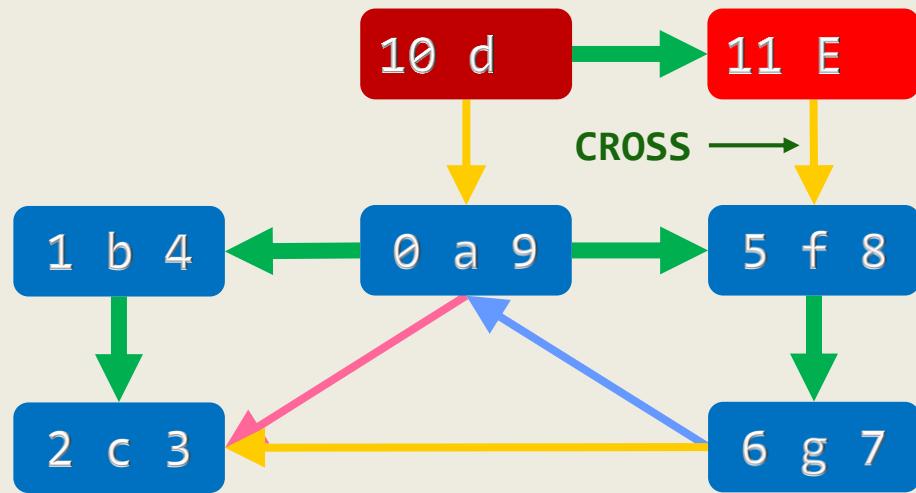
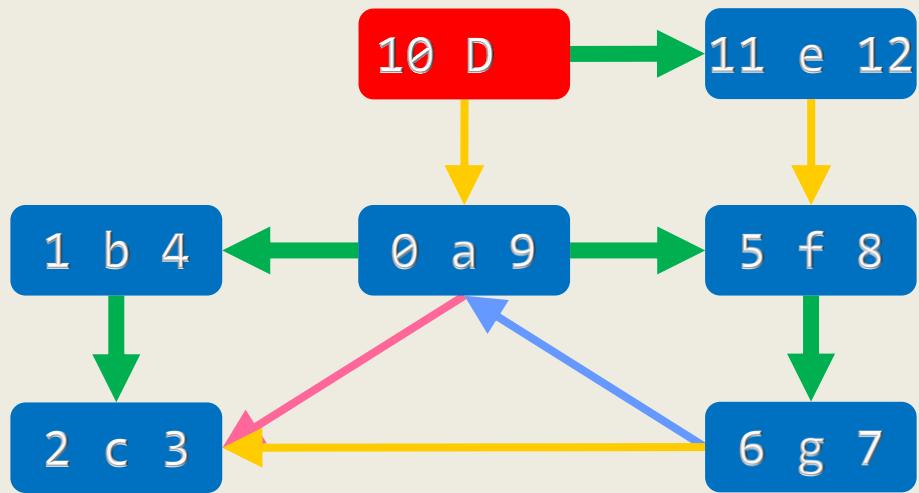


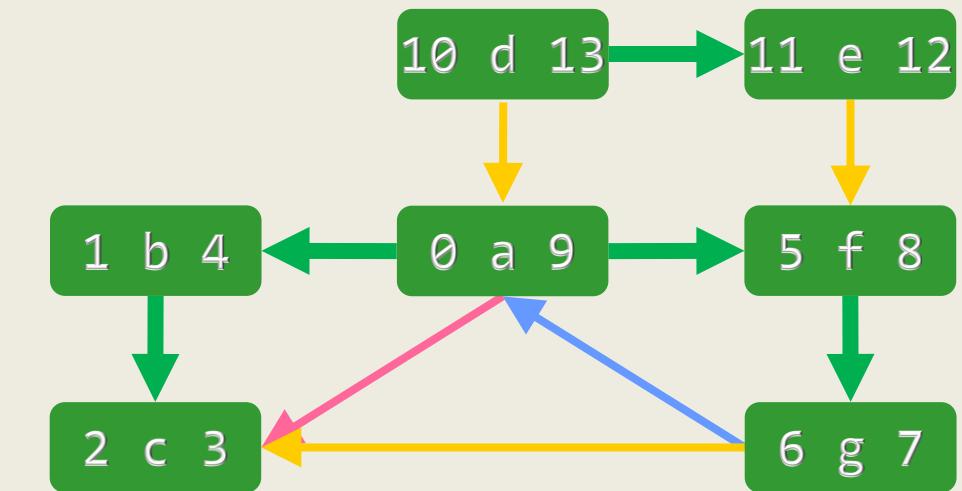
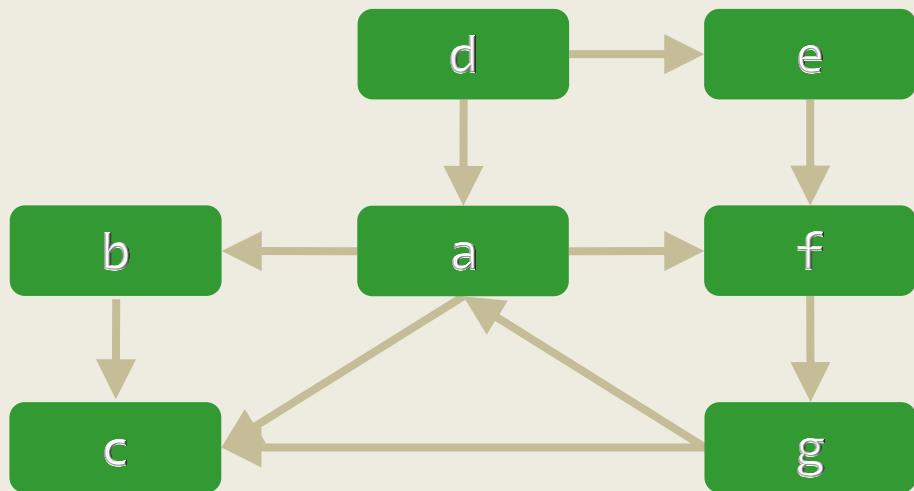
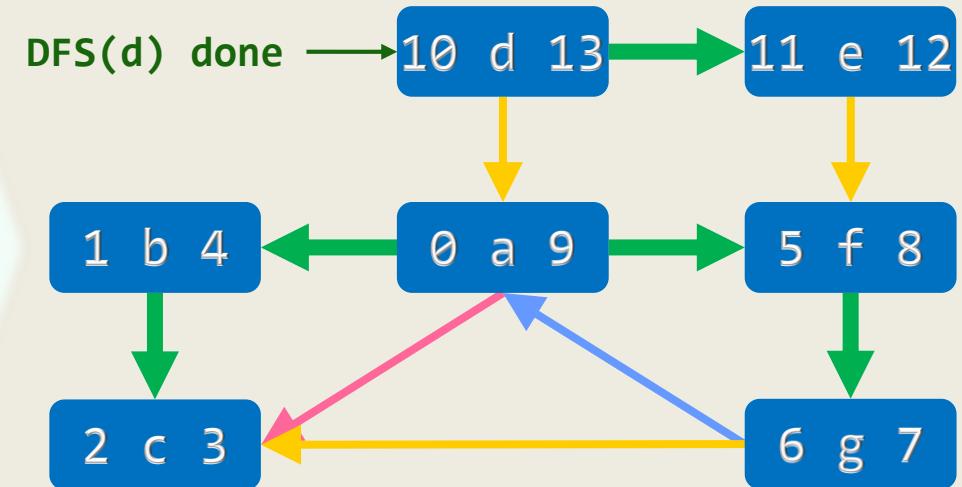
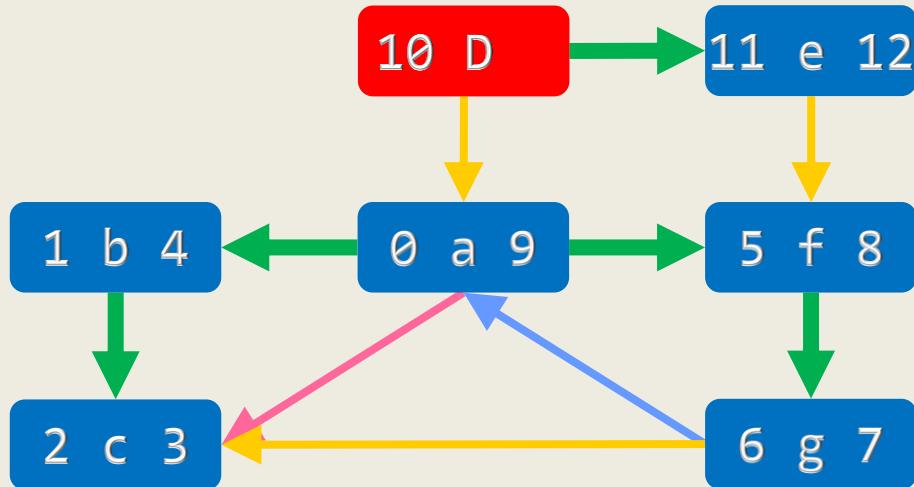












图

# 深度优先搜索：性质

10-E5

邓俊辉

deng@tsinghua.edu.cn

身后有余忘缩手，眼前无路想回头

# 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 (connectivity)
- 在有向图中将访问由s可达的所有顶点 (reachability)

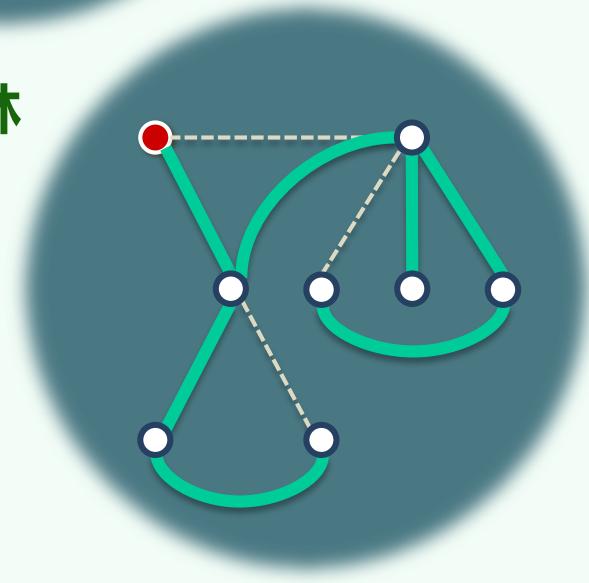
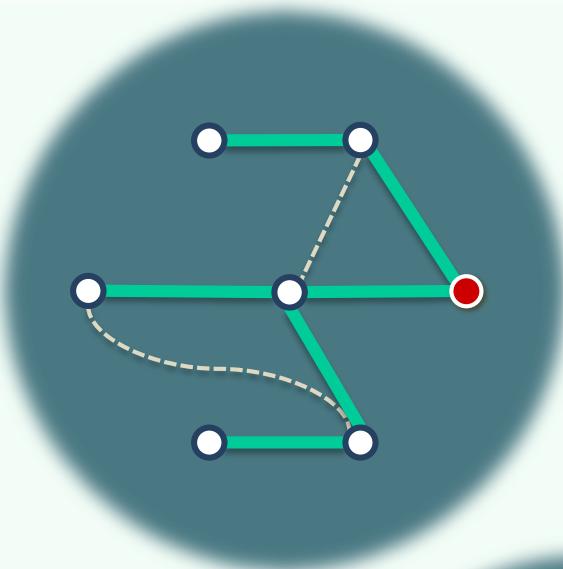
## ❖ 经DFS确定的树边，不会构成回路

## ❖ 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...



# 活跃期 & 括号引理

❖  $active[u] = (dTime[u], fTime[u])$

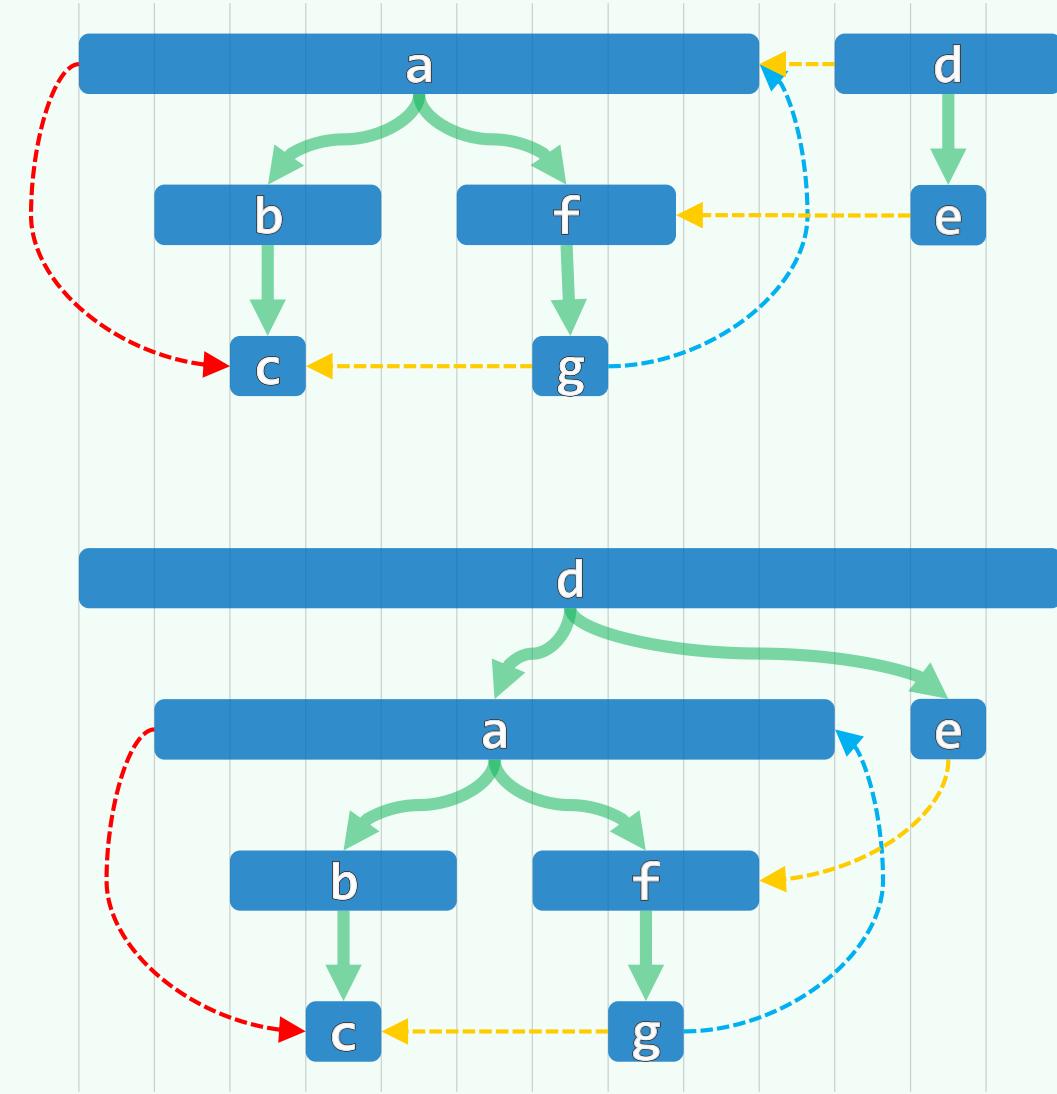
❖ 【Parenthesis Lemma】

给定有向图  $G = (V, E)$  及其任一DFS森林，则

- **u是v的后代 iff  $active[u] \subseteq active[v]$**
- **u是v的祖先 iff  $active[u] \supseteq active[v]$**
- **u与v “无关” iff  $active[u] \cap active[v] = \emptyset$**

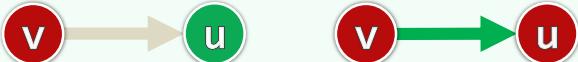
❖ 仅凭  $status[]$ 、 $dTime[]$  和  $fTime[]$

即可对各边分类...



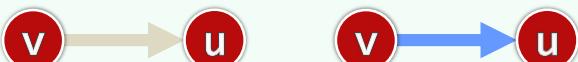
# 边分类

❖ 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 $u$

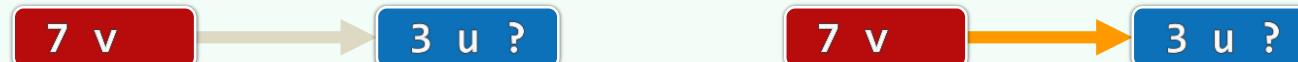
DFS发现后向边 iff 存在回路 //后向边数 == 回路数?

❖ 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

# 遍历算法应用举例

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/二部图判定	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/半径/围长/中心	BFS
欧拉回路	DFS
拓扑排序	DFS
双连通分量、强连通分量分解	DFS
...	...