6.图

（xa）双连通分量

邓俊辉

deng@tsinghua.edu.cn

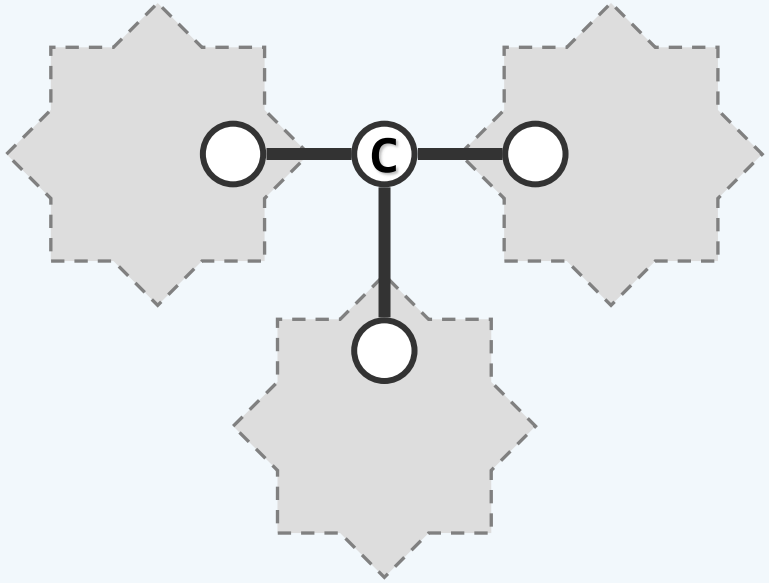
关节点 & 双连通分量

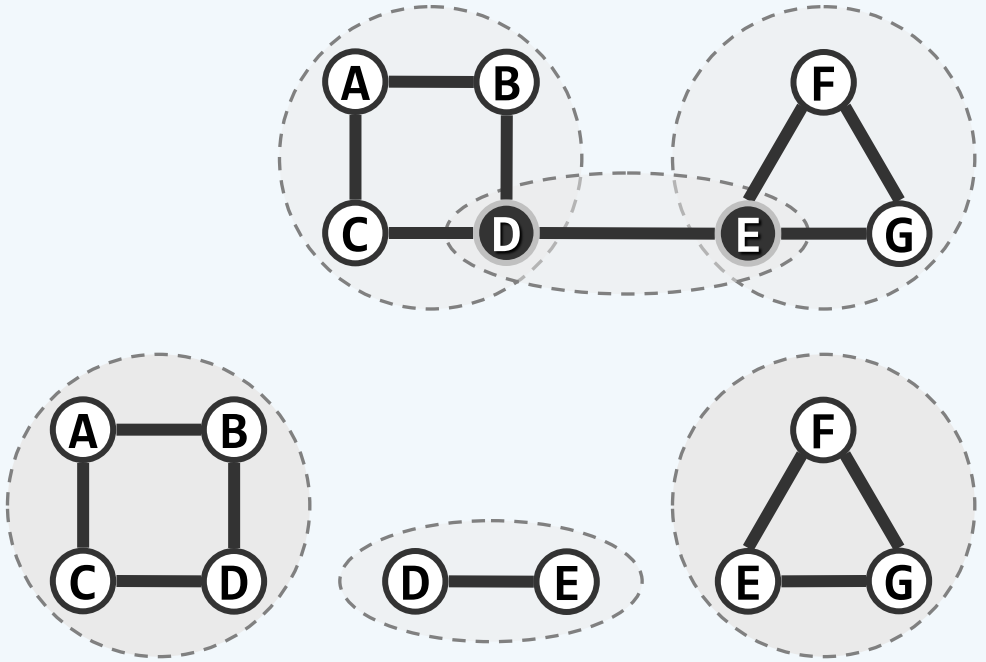
- 无向图的关节点： //articulation point, cut-vertex

  其删除之后，原图的连通分量增多 //connected components

- 无关节点的图，称作双（重）连通图 //bi-connectivity

- 极大的双连通子图，称作双连通分量 //Bi-Connected Components





Brute-Force

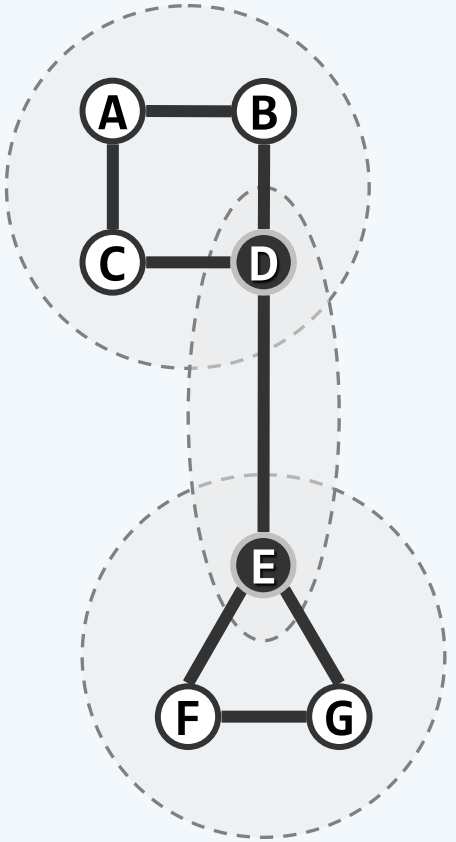
- 给定无向图，如何确定各BCC？
- 先考察简单的版本：如何确定关节点？
- 蛮力：

  对每一顶点v，通过遍历检查G\{v}是否连通
- 共需O(n * (n+e))时间，太慢！

  而且，即便找出关节点，各BCC仍需确定
- 改进：

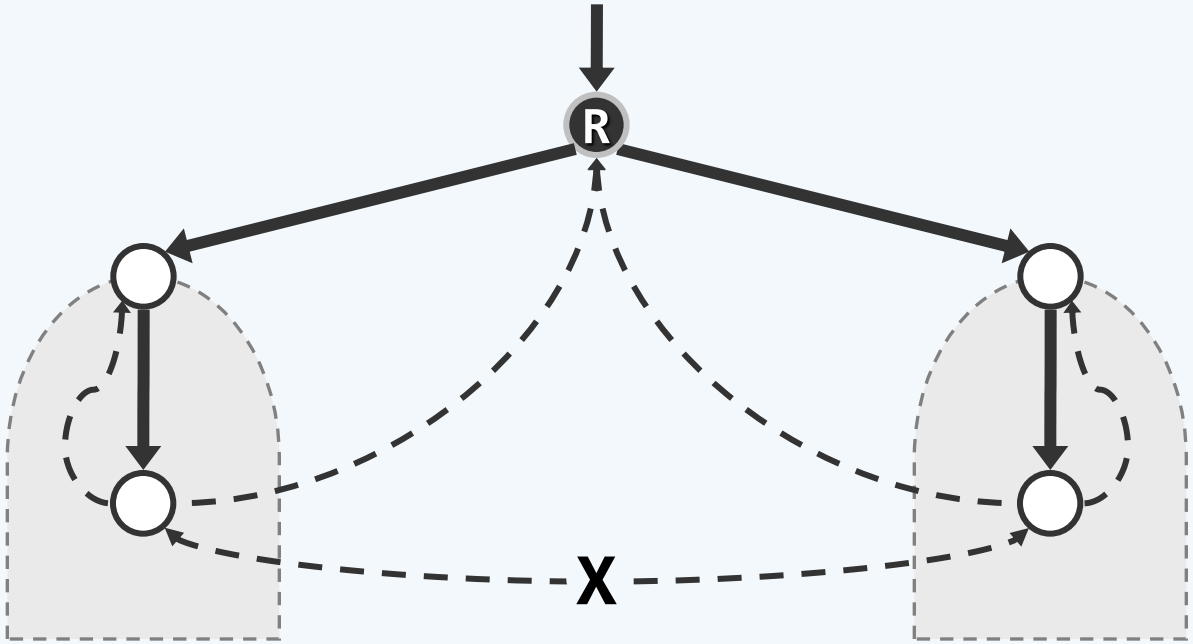
  从任一顶点出发，构造DFS树

  根据DFS留下的标记，甄别是否关节点
- 比如，叶顶点绝不可能是关节点 //为什么？



根顶点

- 根顶点是关节点　iff　树根至少有2棵子树
- 在DFS树中，不难检查从根顶点R发出的树边数目
- 那么，一般的内部顶点呢？



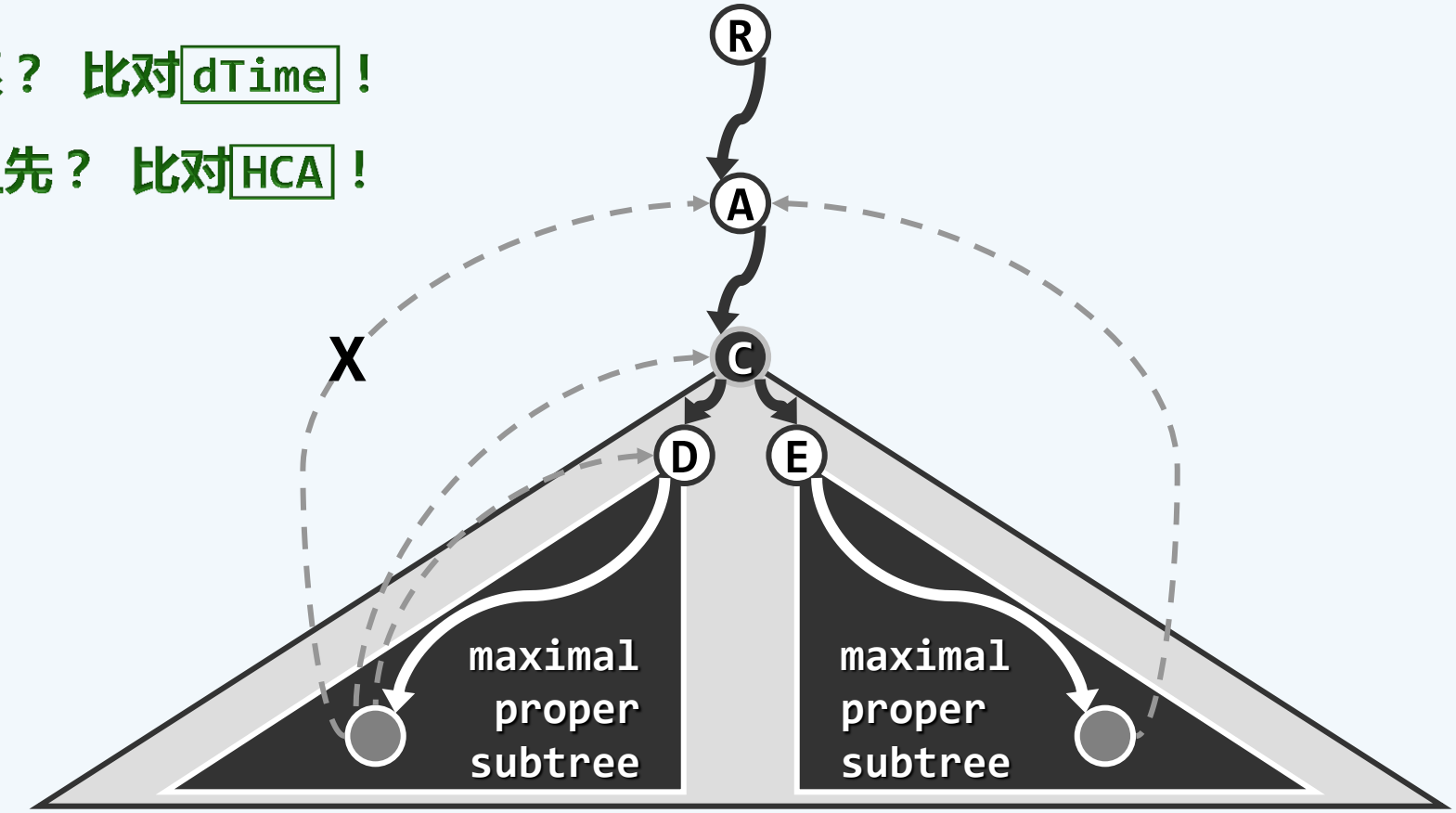
内顶点

❖ 内顶点C是关节点　iff　C的某棵极大真子树与C的真祖先不属于同一BCC

在C的某棵极大真子树中，没有顶点（经后向边）联接到C的真祖先

❖ 如何判断祖先、后代关系？ 比对dTime！

如何记录可联接的最高祖先？ 比对HCA！



Graph::BBC()

```
❖ #define hca(x) ( fTime(x) ) //利用此处闲置的fTime[]充当hca[]
  template <typename Tv, typename Te> //顶点类型、边类型
  void Graph<Tv, Te>::BCC(int v, int& clock, Stack<int>& S) {
     hca(v) = dTime(v) = ++clock;
     status(v) = DISCOVERED; //发现v
     S.push(v); //顶点v入栈，以下枚举v的所有邻居u
     for(int u = firstNbr(v); -1 < u; u = nextNbr(v, u))
        switch ( status(u) )
           { /* ... 视u的状态分别处理 ... */ }
     status(v) = VISITED; //对v的访问结束
  }
  #undef hca
```

switch (status(u))

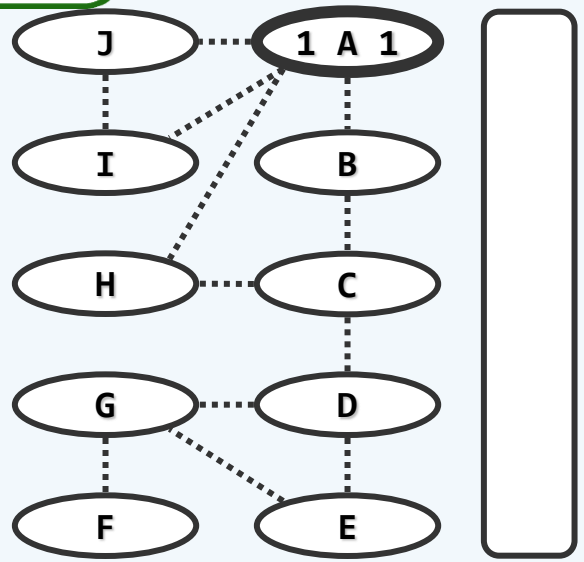
❖ case UNDISCOVERED:

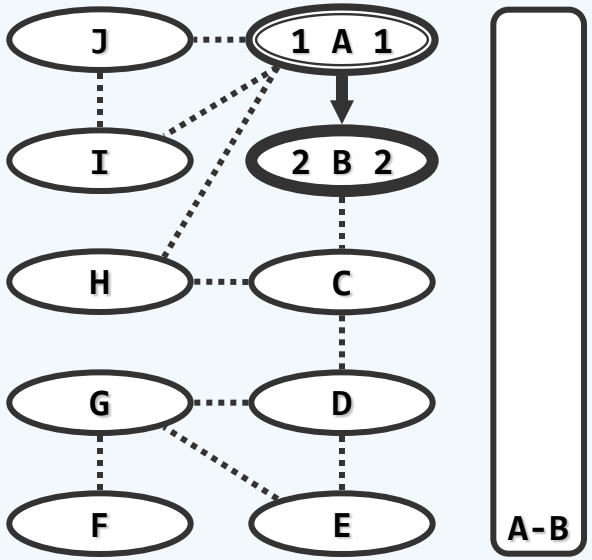
```
parent(u) = v; type(v, u) = TREE; BCC(u, clock, S); //从u开始遍历，返回后
if ( hca(u) < dTime(v) ) { //若u经后向边指向v的真祖先
   hca(v) = min( hca(v), hca(u) ); //则v亦必如此
} else { //否则，以v为关节点
   //u以下即为一个BCC，且其中顶点此时正集中处于栈S的顶部
   //故可依次弹出这些顶点，或根据实际需求转存至其它结构
   while ( v != S.pop() );
   S.push(v); //最后一个顶点（关节点）重新入栈
} //尽管同一顶点可作为（关节点）重复入栈，但至多不过两次
break;
```

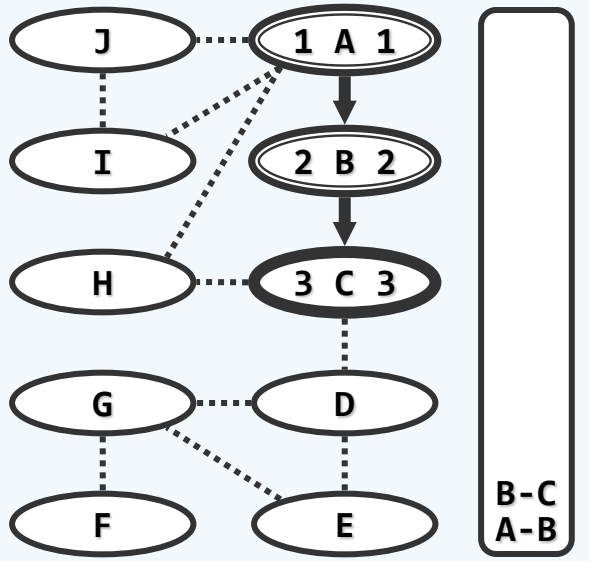
switch (status(u))

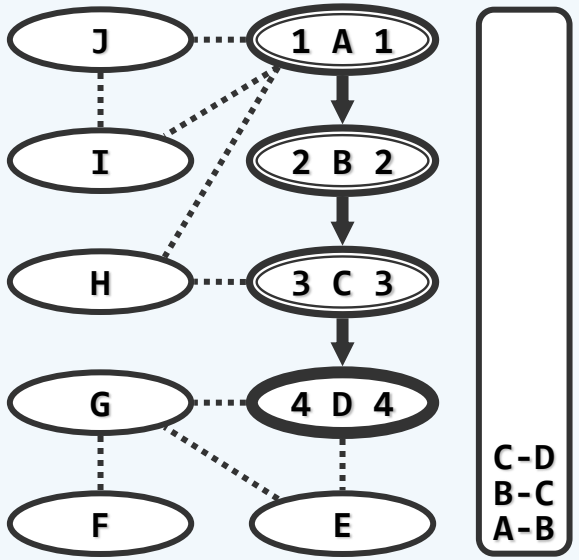
```
❖ case DISCOVERED:
      type(v, u) = BACKWARD;
      if ( u != parent(v) )
         hca(v) = min( hca(v), dTime(u) );
         //更新hca[v]，越小越高
      break;
❖ default: //VISITED (digraphs only)
      type(v, u) =
         dTime(v) < dTime(u) ? FORWARD : CROSS;
      break;
```

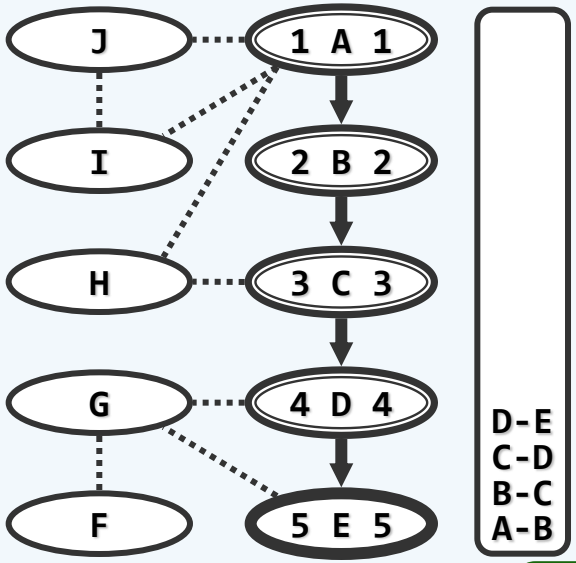
实例

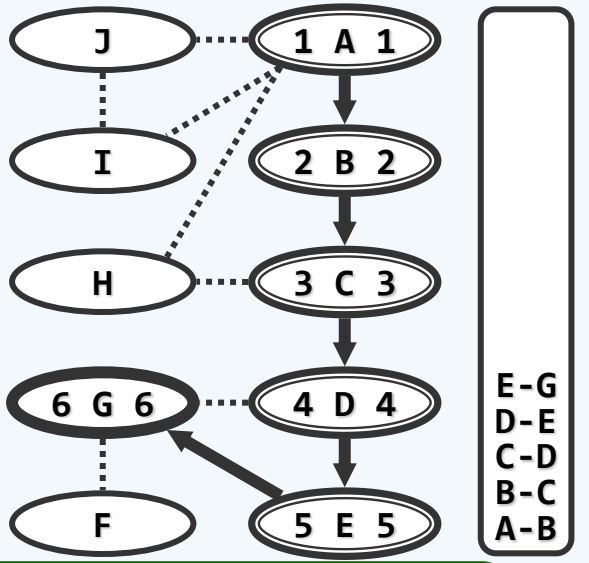

J
1 A 1
I
B
H
C
G
D
F
E


J
1 A 1
I
2 B 2
H
C
G
D
F
E
A-B

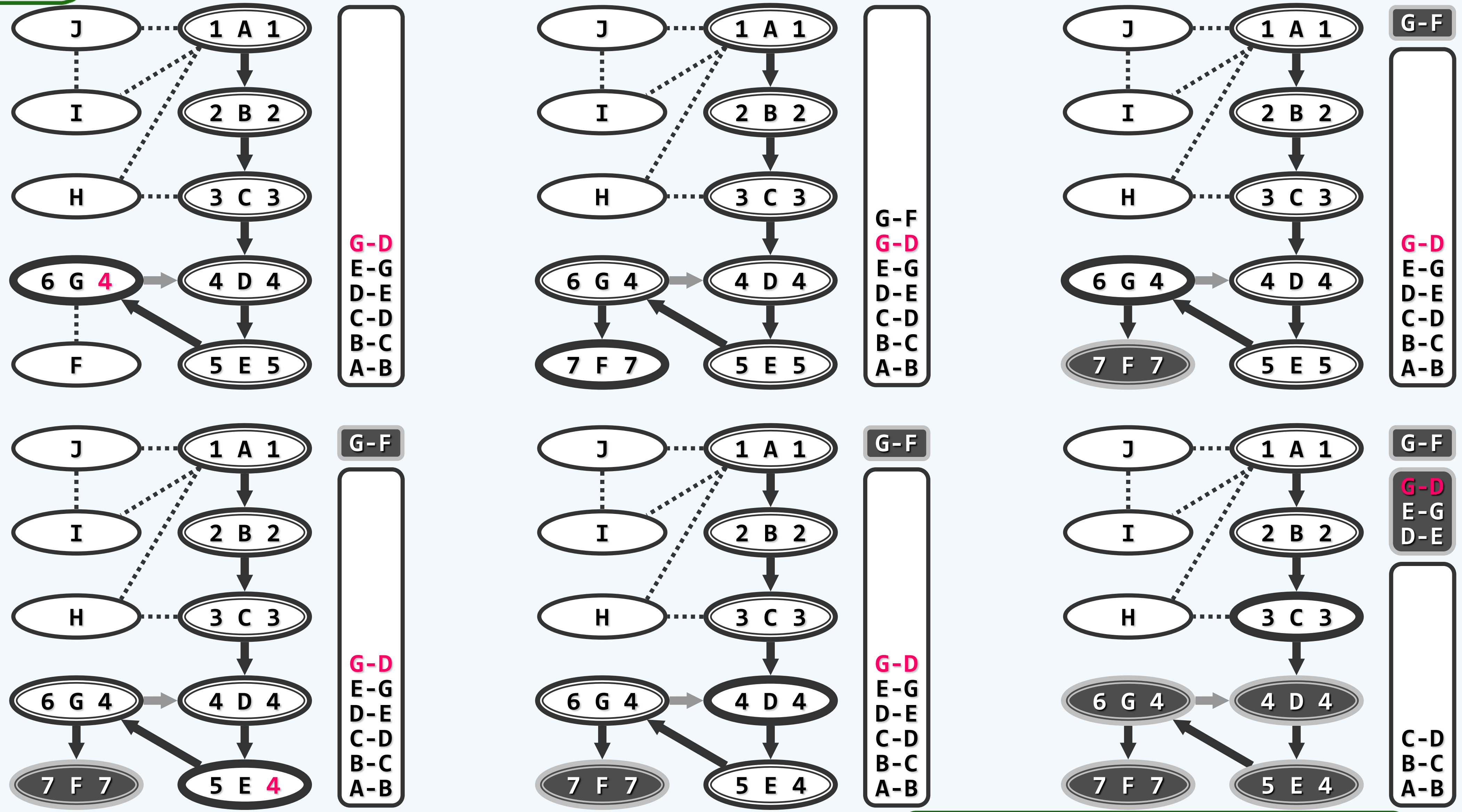

J
1 A 1
I
2 B 2
H
3 C 3
G
D
F
E
B-C
A-B


J
1 A 1
I
2 B 2
H
3 C 3
G
4 D 4
F
E
C-D
B-C
A-B

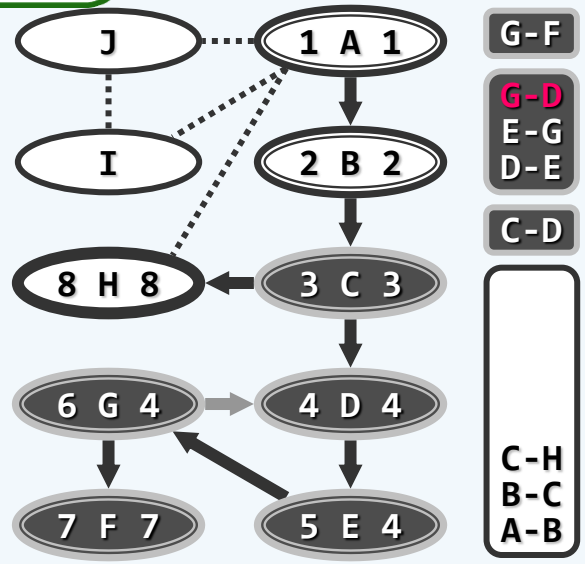

J
1 A 1
I
2 B 2
H
3 C 3
G
4 D 4
F
5 E 5
D-E
C-D
B-C
A-B

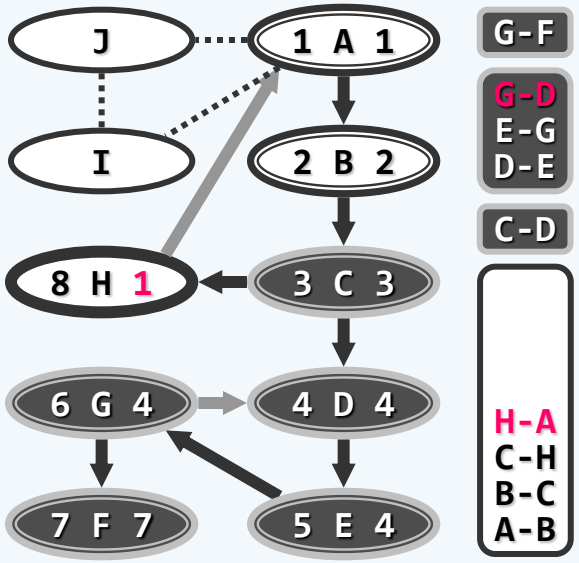

J
1 A 1
I
2 B 2
H
3 C 3
6 G 6
4 D 4
F
5 E 5
E-G
D-E
C-D
B-C
A-B

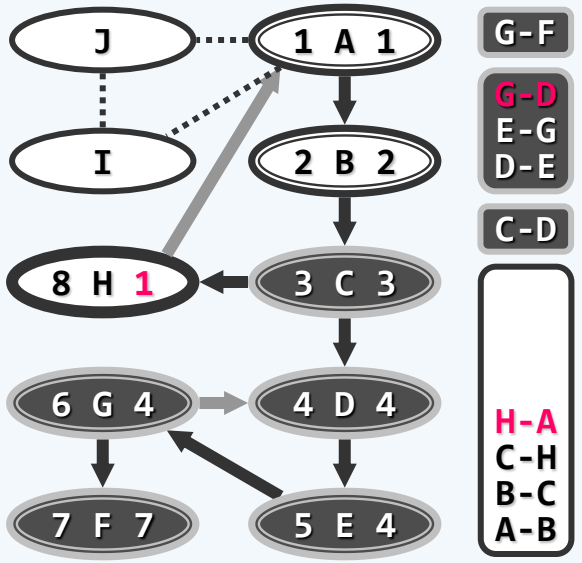
实例

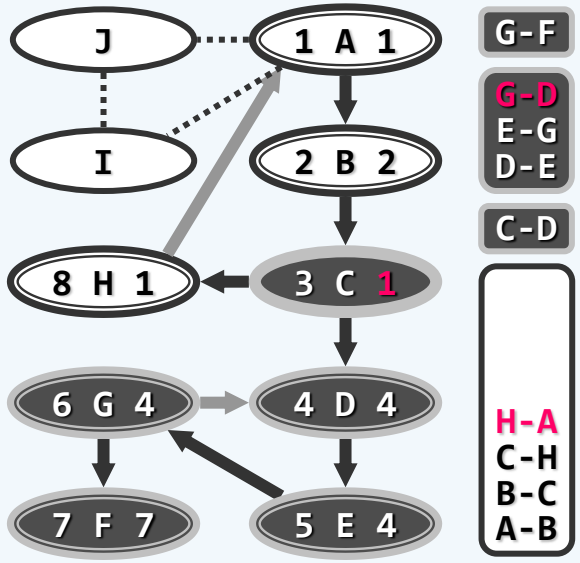


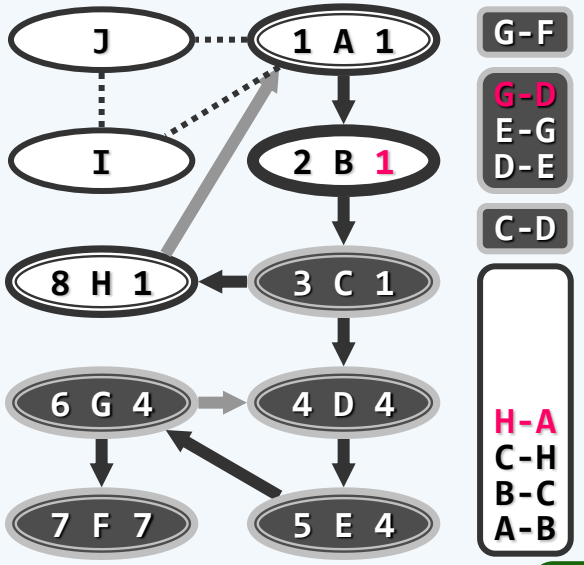
实例

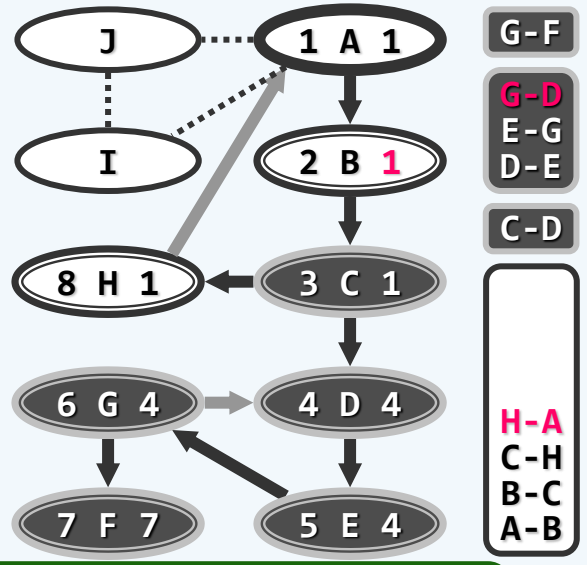

J
1 A 1
I
2 B 2
8 H 8
3 C 3
6 G 4
4 D 4
7 F 7
5 E 4
G-F
G-D
E-G
D-E
C-D
C-H
B-C
A-B


J
1 A 1
I
2 B 2
8 H 1
3 C 3
6 G 4
4 D 4
7 F 7
5 E 4
G-F
G-D
E-G
D-E
C-D
H-A
C-H
B-C
A-B

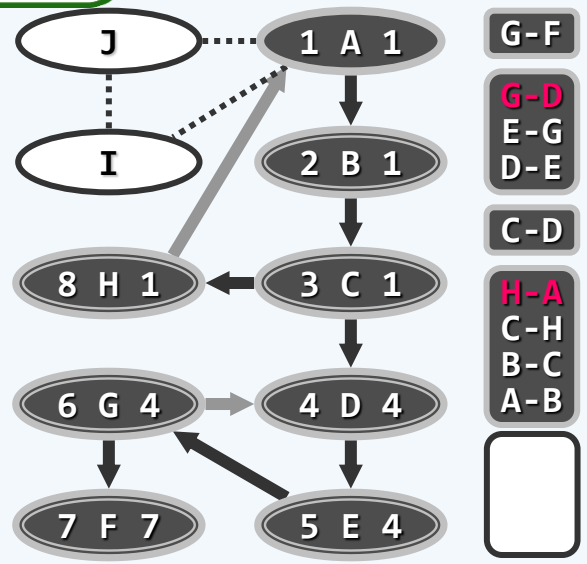

J
1 A 1
I
2 B 2
8 H 1
3 C 3
6 G 4
4 D 4
7 F 7
5 E 4
G-F
G-D
E-G
D-E
C-D
H-A
C-H
B-C
A-B

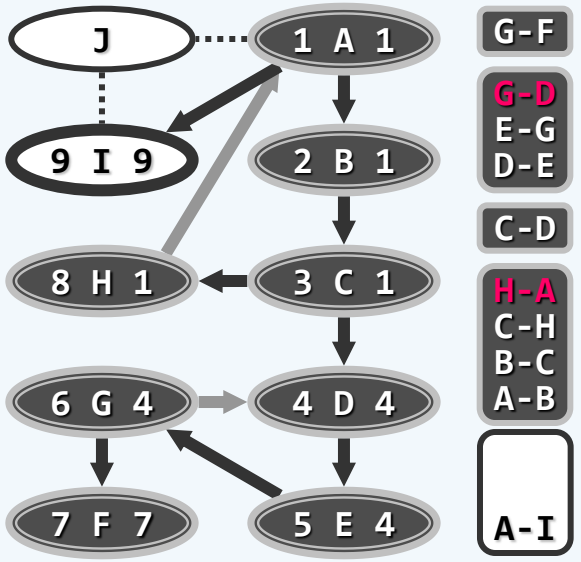

J
1 A 1
I
2 B 2
8 H 1
3 C 1
6 G 4
4 D 4
7 F 7
5 E 4
G-F
G-D
E-G
D-E
C-D
H-A
C-H
B-C
A-B

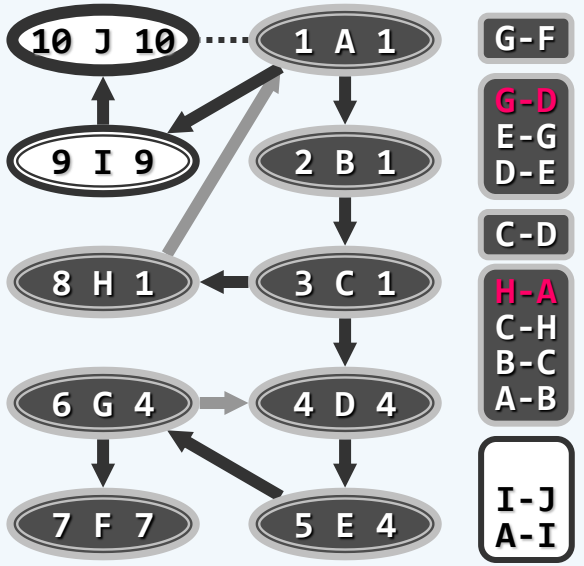

J
1 A 1
I
2 B 1
8 H 1
3 C 1
6 G 4
4 D 4
7 F 7
5 E 4
G-F
G-D
E-G
D-E
C-D
H-A
C-H
B-C
A-B

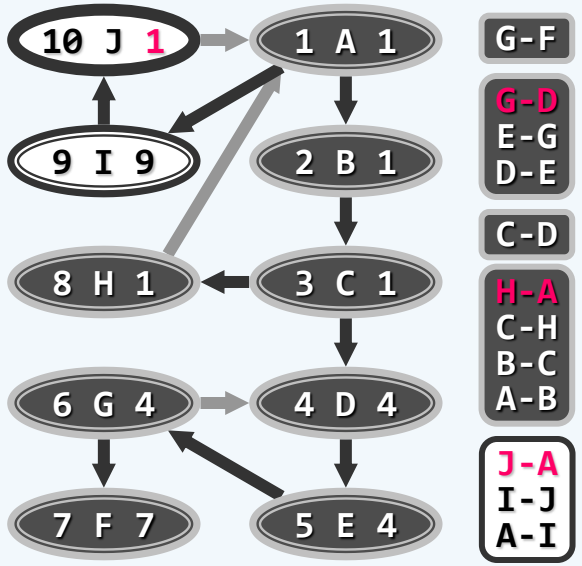

J
1 A 1
I
2 B 1
8 H 1
3 C 1
6 G 4
4 D 4
7 F 7
5 E 4
G-F
G-D
E-G
D-E
C-D
H-A
C-H
B-C
A-B

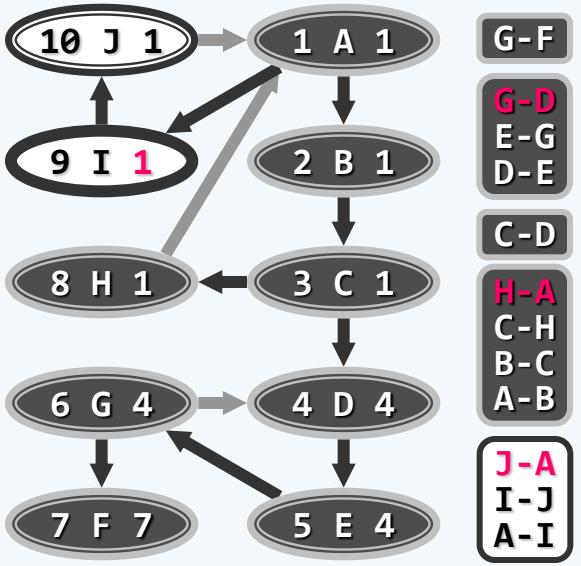
实例

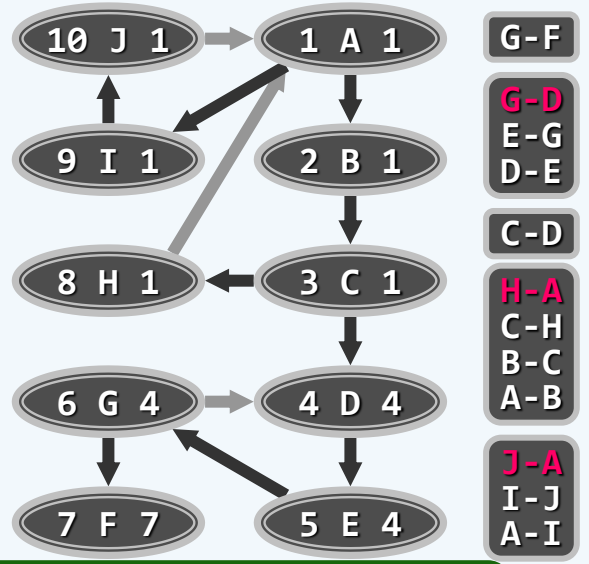

J
1 A 1
I
2 B 1
8 H 1
3 C 1
6 G 4
4 D 4
7 F 7
5 E 4
G-F
G-D
E-G
D-E
C-D
H-A
C-H
B-C
A-B


J
1 A 1
9 I 9
2 B 1
8 H 1
3 C 1
6 G 4
4 D 4
7 F 7
5 E 4
G-F
G-D
E-G
D-E
C-D
H-A
C-H
B-C
A-B
A-I


10 J 10
1 A 1
9 I 9
2 B 1
8 H 1
3 C 1
6 G 4
4 D 4
7 F 7
5 E 4
G-F
G-D
E-G
D-E
C-D
H-A
C-H
B-C
A-B
I-J
A-I


10 J 1
1 A 1
9 I 9
2 B 1
8 H 1
3 C 1
6 G 4
4 D 4
7 F 7
5 E 4
G-F
G-D
E-G
D-E
C-D
H-A
C-H
B-C
A-B
J-A
I-J
A-I


10 J 1
1 A 1
9 I 1
2 B 1
8 H 1
3 C 1
6 G 4
4 D 4
7 F 7
5 E 4
G-F
G-D
E-G
D-E
C-D
H-A
C-H
B-C
A-B
J-A
I-J
A-I


10 J 1
1 A 1
9 I 1
2 B 1
8 H 1
3 C 1
6 G 4
4 D 4
7 F 7
5 E 4
G-F
G-D
E-G
D-E
C-D
H-A
C-H
B-C
A-B
J-A
I-J
A-I

复杂度

- 运行时间与常规的DFS相同，也是$O(n + e)$

  自行验证：栈操作的复杂度也不过如此

- 除原图本身，还需一个容量为$O(e)$的栈存放已访问的边

  为支持递归，另需一个容量为$O(n)$的运行栈

- 课后：

  该算法是否也适用于非连通图？

  在有向图中如何实现对应的计算？

- Strongly-connected component

  Kosaraju's algorithm

  Tarjan's algorithm