

Introduction to

Algorithm Design and Analysis

[13] Undirected Graph

Jingwei Xu

[https://cs.nju.edu.cn/ics/people/jingweixu/
index.html](https://cs.nju.edu.cn/ics/people/jingweixu/index.html)

Institute of Computer Software
Nanjing University

In the last class ...

- **Directed Acyclic Graph**
 - Topological order
 - Critical path analysis
- **Strongly Connected Component (SCC)**
 - Strong connected component and condensation
 - Finding SCC based on DFS

DFS on Undirected Graph

- **Undirected Graph**

- Symmetric Digraph
- Undirected Graph DFS Skeleton

- **Biconnected Components**

- Articulation Points
- Bridge

- **Other undirected graph problems**

- Orientation of an undirected graph
- Simplified Minimum Spanning Tree

What is Different for “Undirected”

- Characteristics of undirected graph traversal
 - One edge may be traversed for **two times** in opposite directions.
- For an undirected graph, DFS provides an orientation for each of its edges
 - Oriented in the direction in which they are first encountered.

Edges in DFS

- **Cross edge**

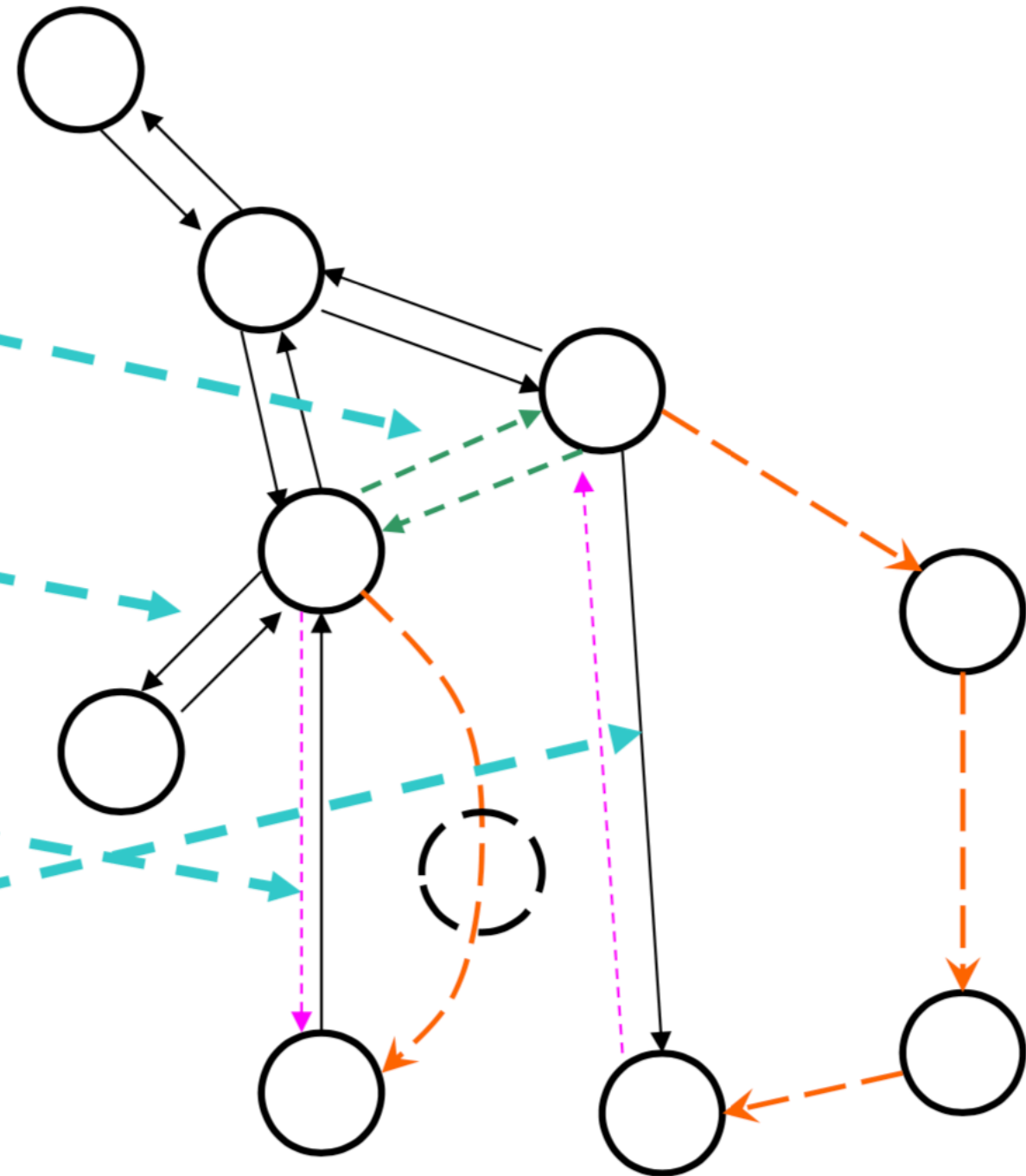
- Not existing

- **Back edge**

- Back to the direct parent:
second encounter
- Otherwise: **first encounter**

- **Forward edge**

- Always **second encounter, and first time as back edge**



Modifications to the DFS Skeleton

- All the **second encounter** are **bypassed**.
- So, the **only substantial modification** is for the possible back edges leading to an ancestor, but not direct parent.
- We need know the **parent**, that is, the direct ancestor, for the vertex to be processed.

DFS Skeleton for Undirected Graph

- `void dfsSweep(intList[] adjVertices, int n, ...)`
- `int ans;`
- `<Allocate color array and initialize to white>`
- `for each vertex v of G, in some order`
- `if(color[v]==white)`
- `Int vAns=dfs(adjVertices, color, v, -1,...);`
- `<Process vAns>`
- `//continue loop`
- `return ans;`

DFS Skeleton for Undirected Graph

- `int dfs(intList[] adjVertices, int[] color, int v, int p,...)`
- `int w; intList remAdj; int ans; color[v]=gray;`
- `<Preorder processing of vertex v>`
- `remAdj=adjVertices[v];`
- `while(remAdj != nil)`
- `w=first(remAdj);`
- `if(color[w]==white)`
- `<Exploratory processing for tree edge vw>`
- `dfs(adjVertices, color, w, v, ...);`
- `<Backtrack processing for tree edge vw, using wAns>`
- `else if(color[w]==gray && w!=p)`
- `<Checking for nontree edge vw>`
- `remAdj=rest(remAdj);`
- `<Postorder processing of vertex v, including final computation of ans>`
- `color[v]=black;`
- `return ans;`

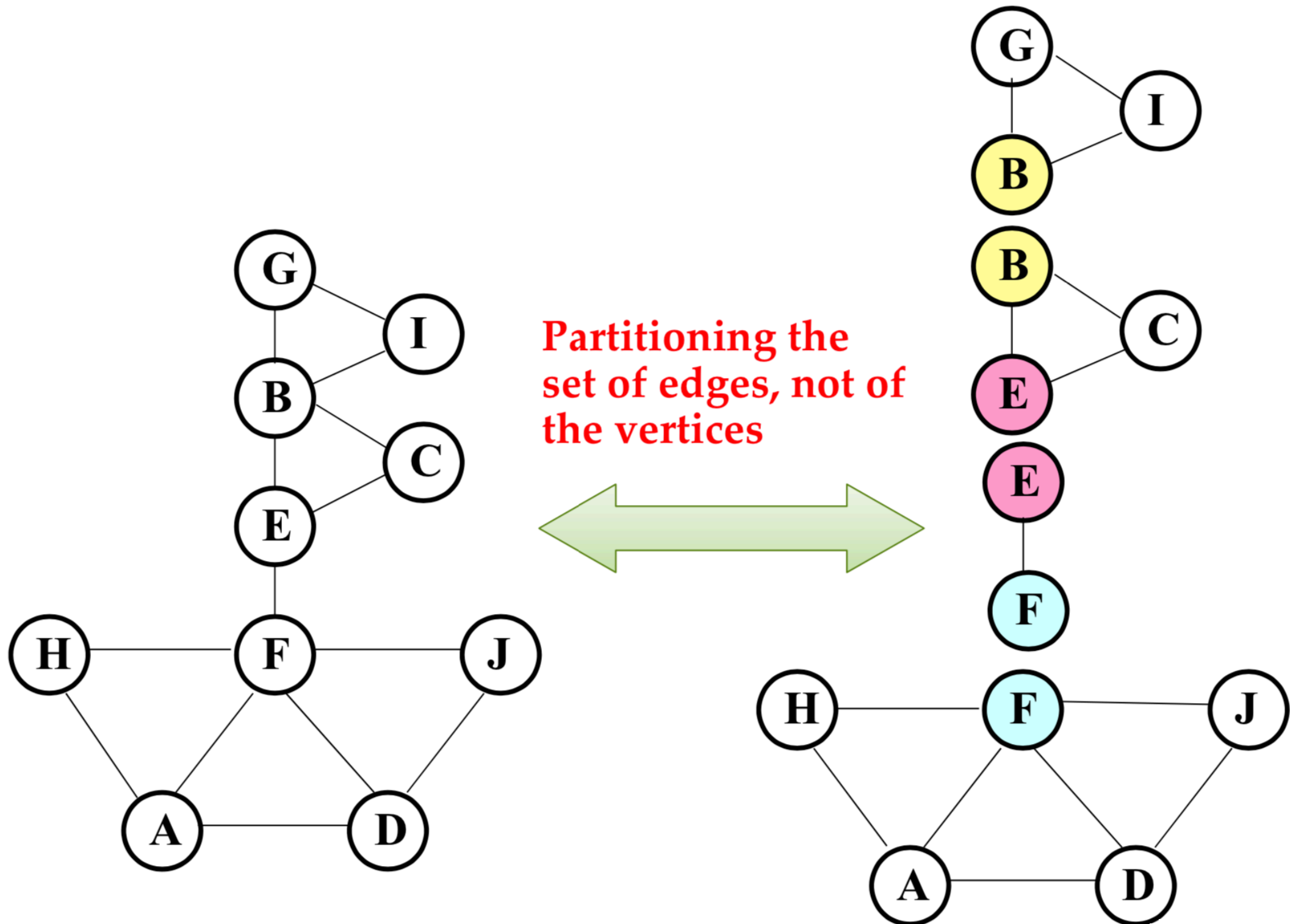
Complexity of Undirected DFS

- $\Theta(m+n)$
 - If each inserted statement for specialized application runs in constant time
 - The same with directed graph DFS
- Extra space $\Theta(n)$
 - For array color, or activation frames of recursion

Biconnected Graph

- Being connected
 - Tree: acyclic, least (cost) connected
 - Node/edge connected: fault-tolerant connection
- Articulation point (2-node connected)
 - v is an articulation point if deleting v leads to disconnection
- Bridge (2-edge connected)
 - uv is a bridge if deleting uv leads to disconnection

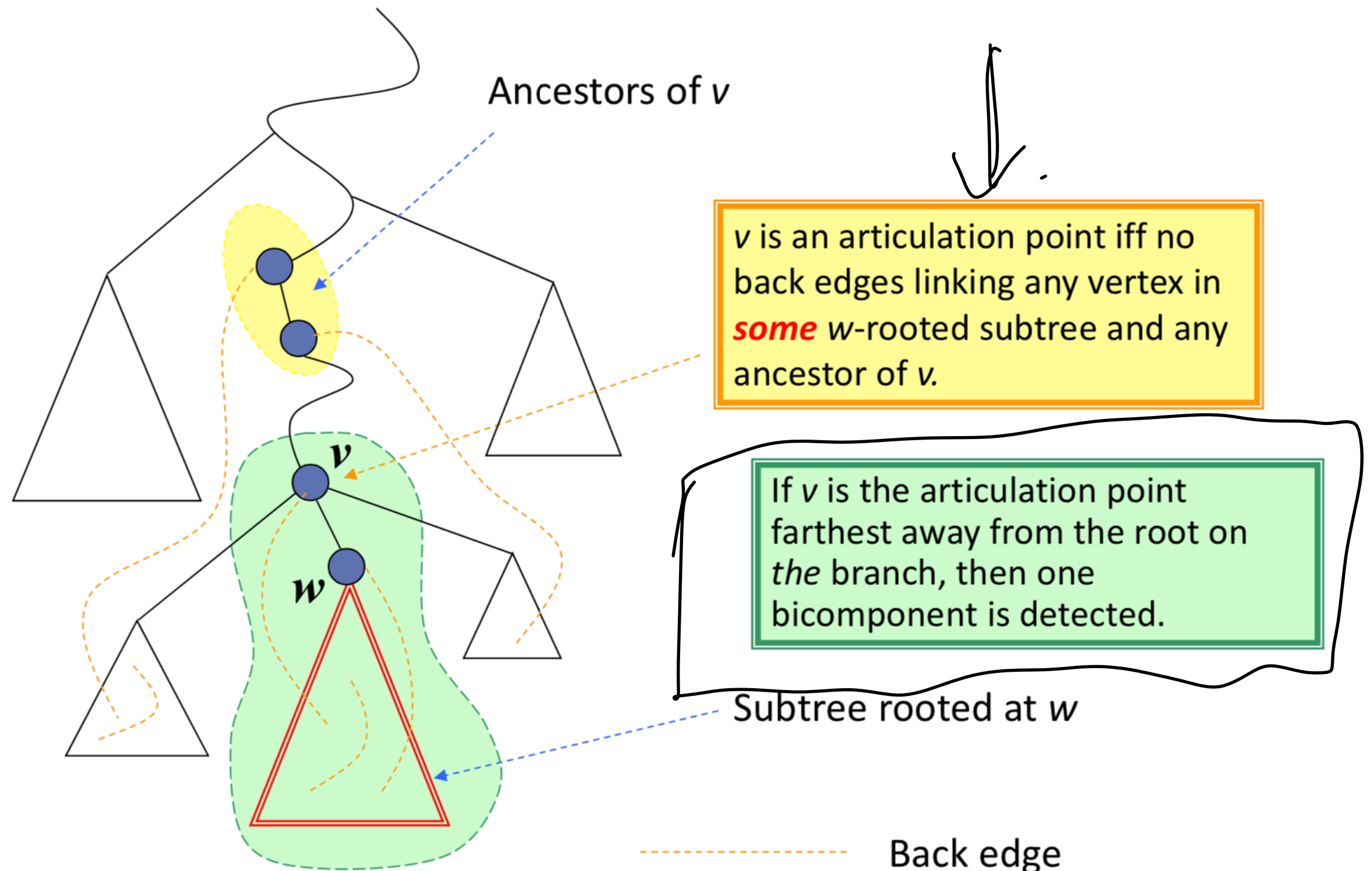
Articulation Points



Definition Transformation

- “Short definition”
 - Deleting v leads to disconnection
- “Long definition”
 - If there **exist** nodes w and x , such that v is in **every** path from w to x (w and x are vertices different from v)
- “Long definition” or “DFS definition”
 - **No** back edges linking **any** vertex in **some** w -rooted subtree and any ancestor of v

Articulation Point Algorithm

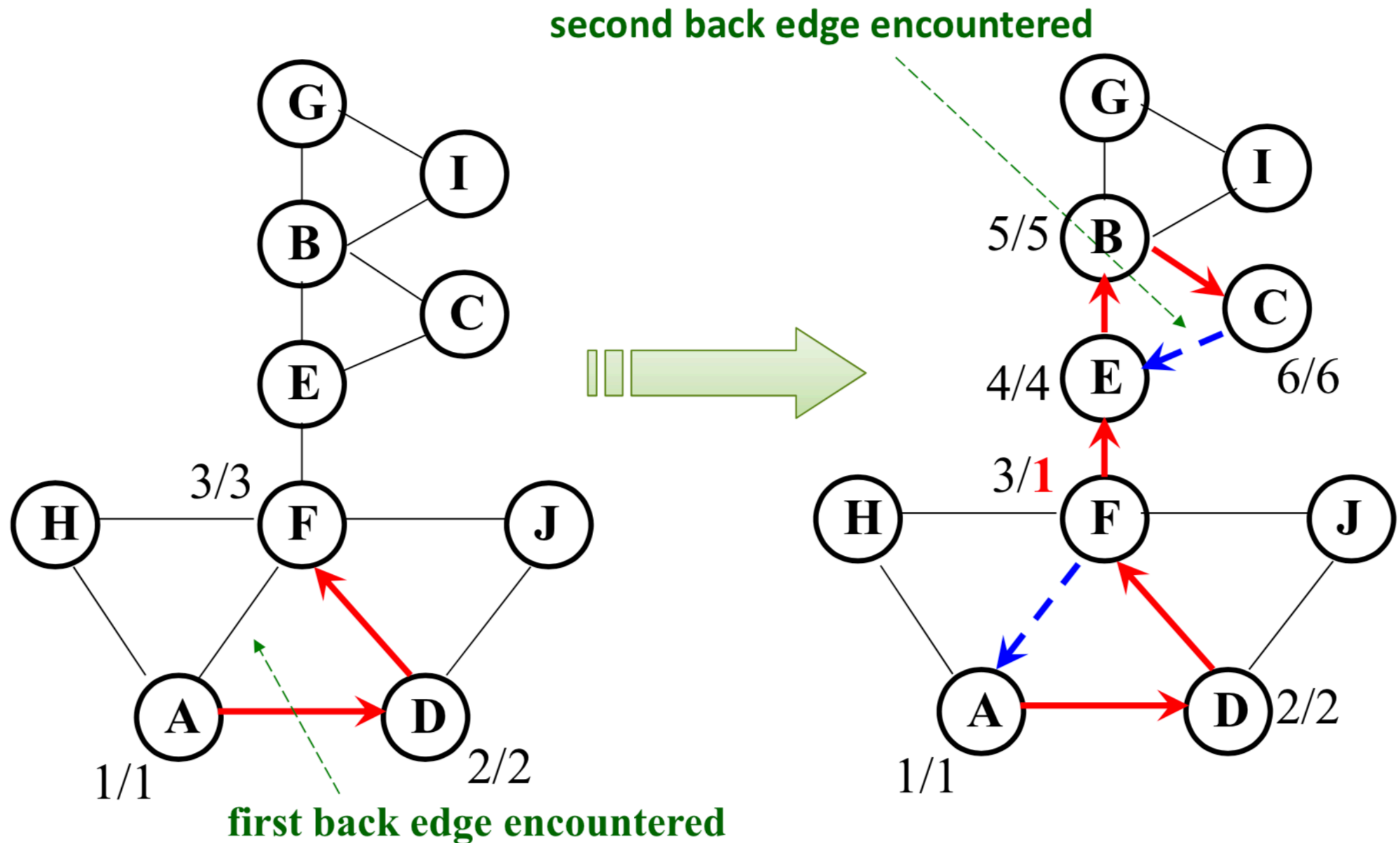


Updating the value of **back**

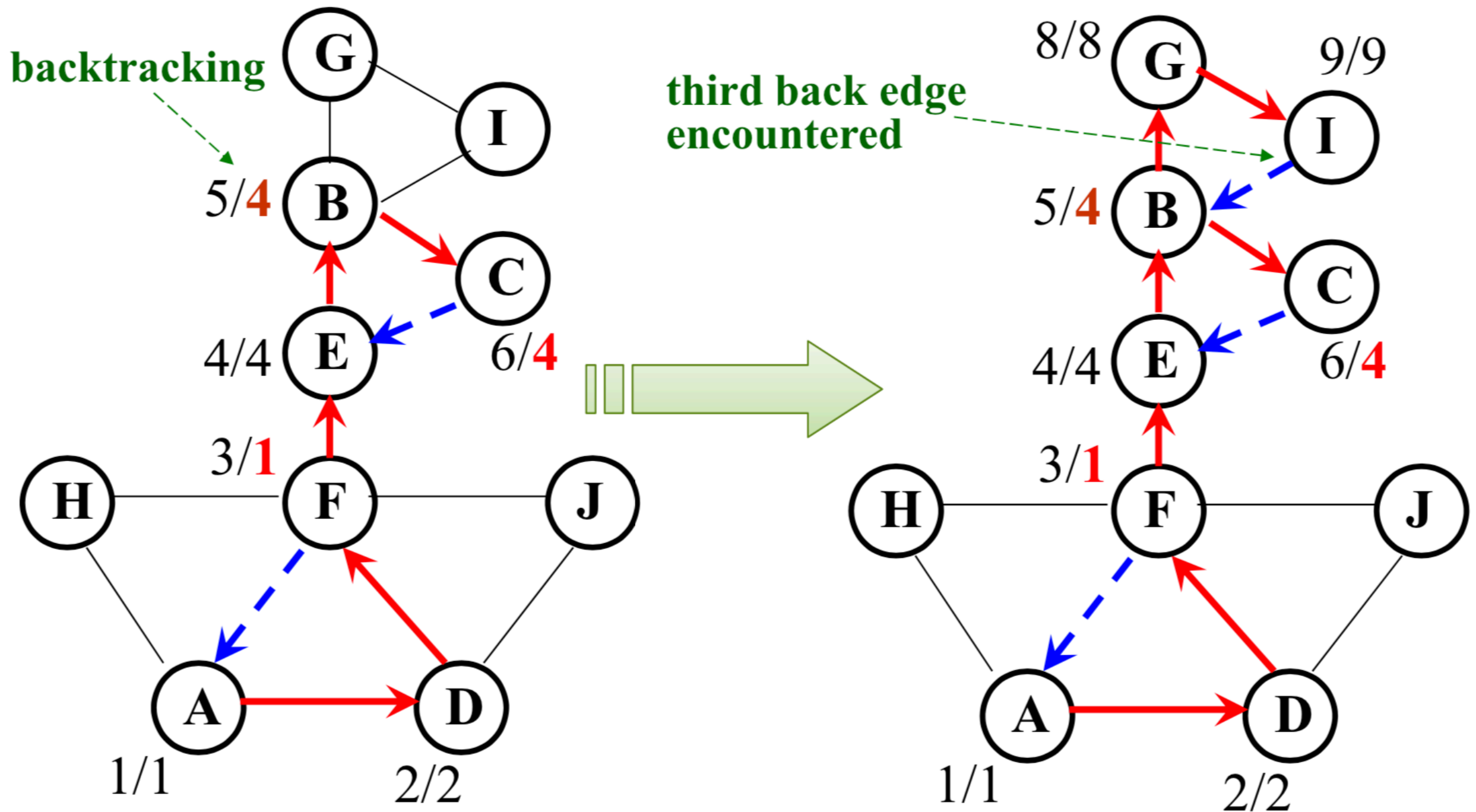
- **v first discovered**
 - $\text{back} = \text{discoverTime}(v)$
- **Trying to explore, but a back edge vw from v encountered**
 - $\text{back} = \min(\text{back}, \text{discoverTime}(w))$
- **Backtracking from w to v**
 - $\text{back} = \min(\text{back}, \text{wback})$

The back value of v is the smallest discover time a back edge “sees” from **any** subtree of v .

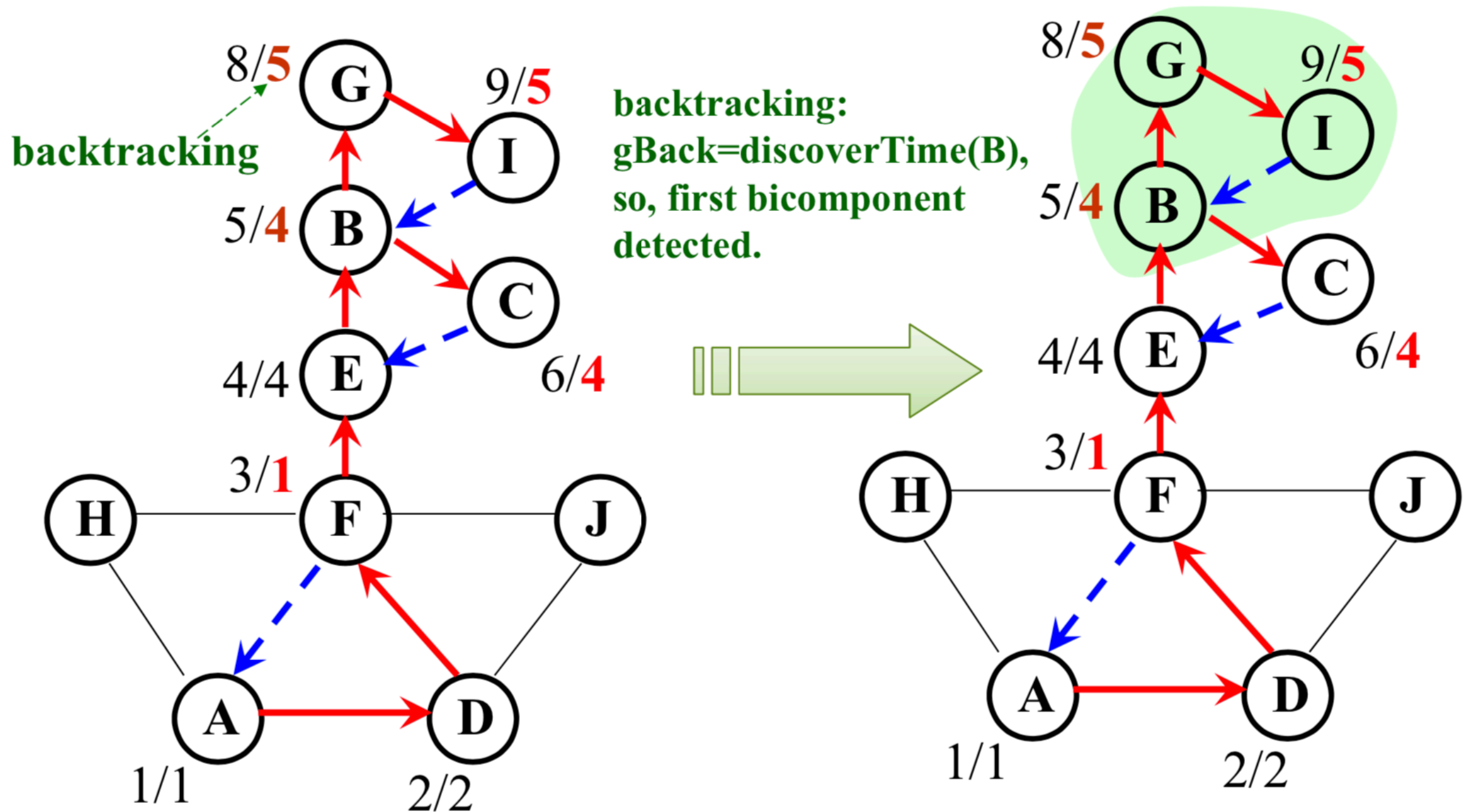
Example



Example



Example



Keeping the Track of Backing

- Tracking data

- For each vertex v , a local variable $back$ is used to store the required information, as the value of **discoverTime** of some vertex.

- Testing for bicomponent

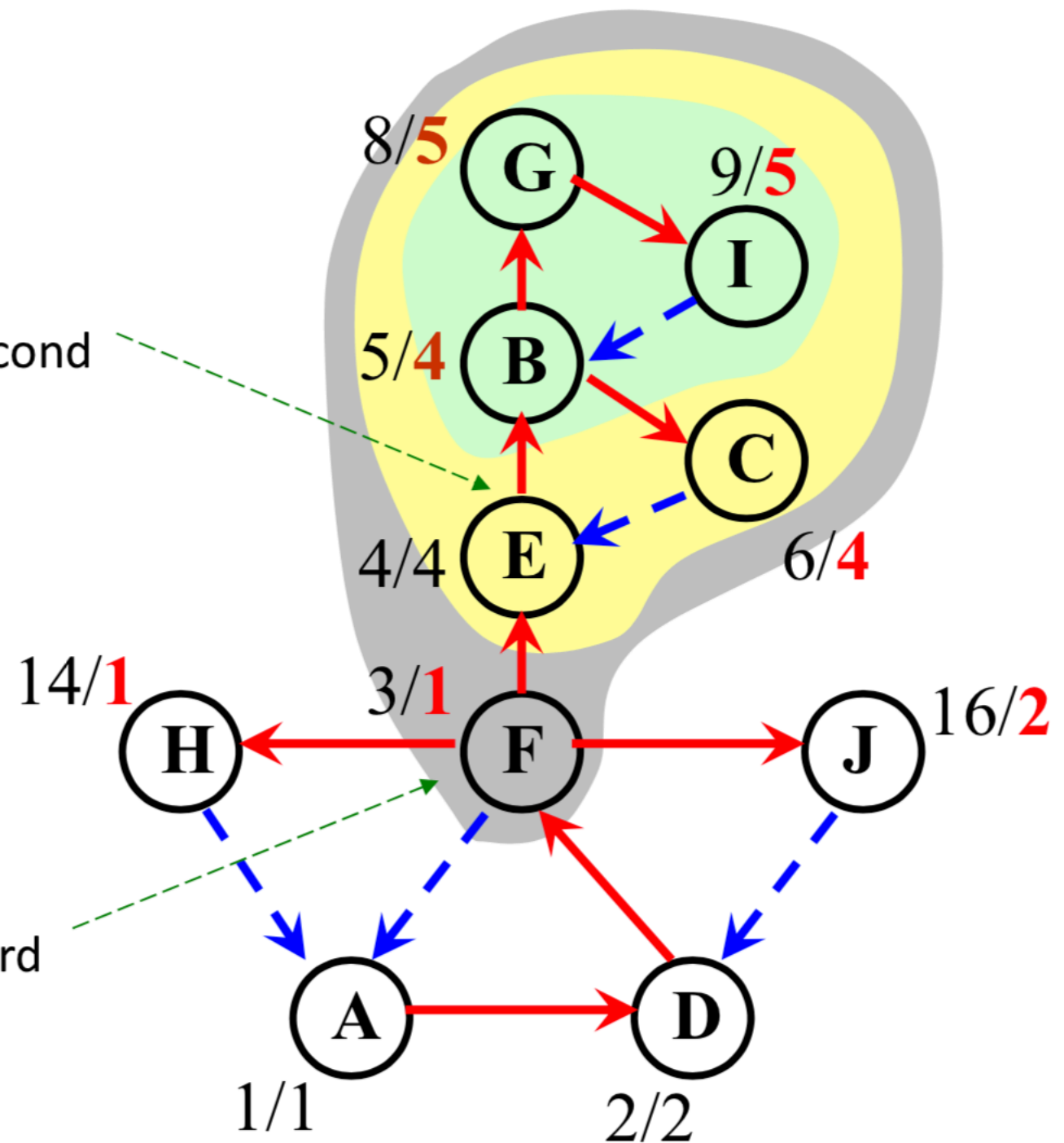
- At backtracking from w to v , the condition implying a bicomponent is:
 - $wBack \geq discoverTime(v)$
(where **wback** is the returned back value for w)

When $back$ is no less than the discover time of v , there is at least one subtree of v connected to other part of the graph only by v .

Example

Backtracking from B to E:
bBack=discoverTime(E), so, the second
bicomponent is detect

Backtracking from E to F:
eBack>discoverTime(F), so, the third
bicomponent is detect



Articulation Point Algorithm

Algorithm 12: ARTICULATION-POINT-DFS(v)

```
1  $v.color := \text{GRAY}$  ;
2  $time := time + 1$  ;
3  $v.discoverTime := time$  ;
4  $v.back := v.discoverTime$  ;
5 foreach neighbor  $w$  of  $v$  do
6     if  $w.color = \text{WHITE}$  then
7          $w.back := \text{ARTICULATION-POINT-DFS}(w)$  ;
8         if  $w.back \geq v.discoverTime$  then
9             Output  $v$  as an articulation point ;
10         $v.back := \min\{v.back, w.back\}$  ;
11    else
12        if  $vw$  is  $BE$  then                                     /*  $w$  是  $v$  非父节点的祖先节点 */
13             $v.back := \min\{v.back, w.discoverTime\}$  ;
14 return  $back$  ;
```

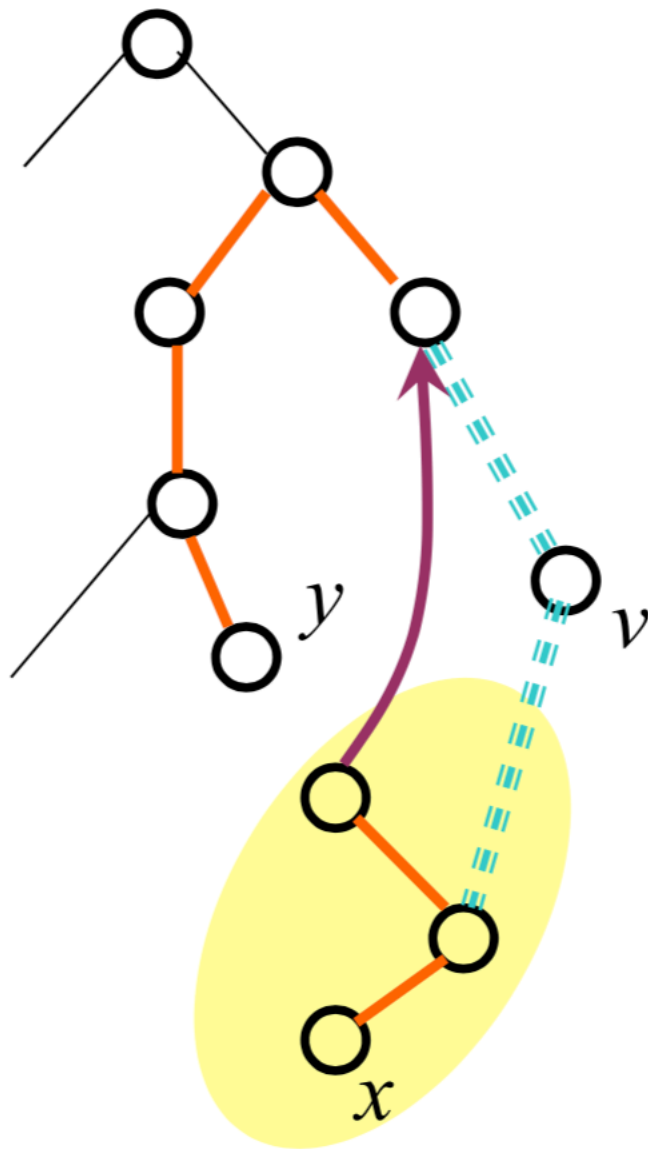
Correctness

- We have seen that:
 - If v is the articulation point farthest away from the root on the branch, then one bicomponent is detected.
- So, we need only prove that:
 - In a DFS tree, a vertex (not root) v is an articulation point **if and only if** (1) v is not a leaf; (2) **some** subtree of v has **no back edge** incident with a proper ancestor of v .

Characteristics of Articulation Point

- In a DFS tree, a vertex (not root) v is an articulation point **if and only if** (1) v is not a leaf; (2) **some** subtree of v has **no back edge** incident with a proper ancestor of v .
- \Leftarrow Trivial
- \Rightarrow
 - By definition, v is on every path between some x, y (different from v).
 - At least one of x, y is a proper descendent of v (otherwise, $x \leftrightarrow \text{root} \leftrightarrow y$ not containing v).
 - By **contradiction**, suppose that every subtree of v has a back edge to a proper ancestor of v , we can find a xy -path not containing v for all possible cases (only 2 cases)

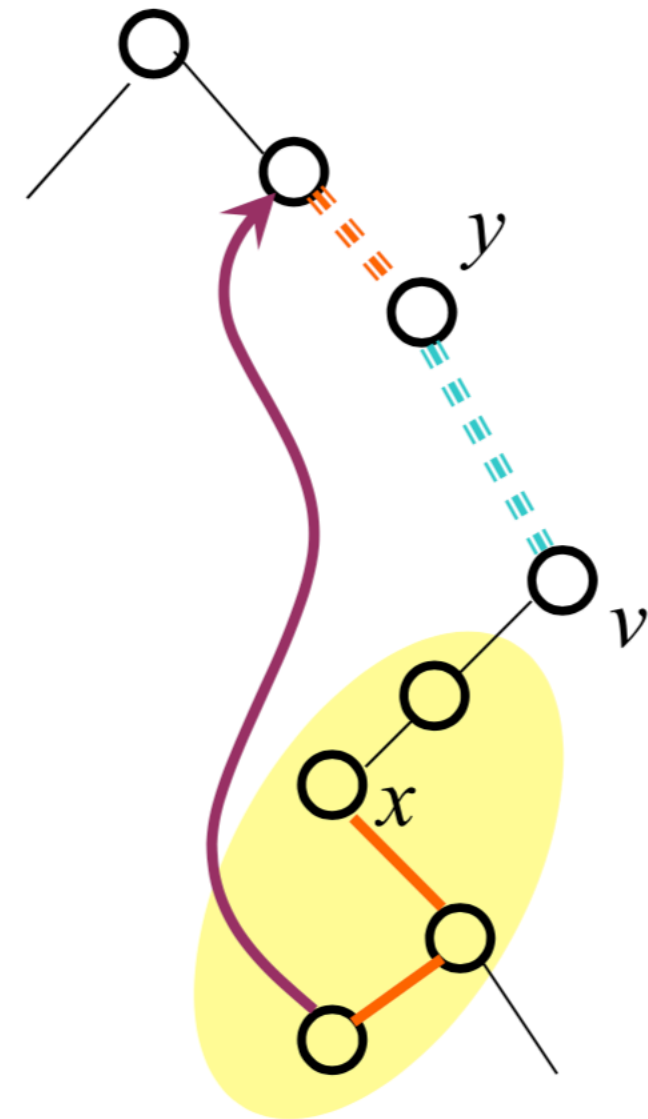
Case 1



Case 1.1: another is not an ancestor of v

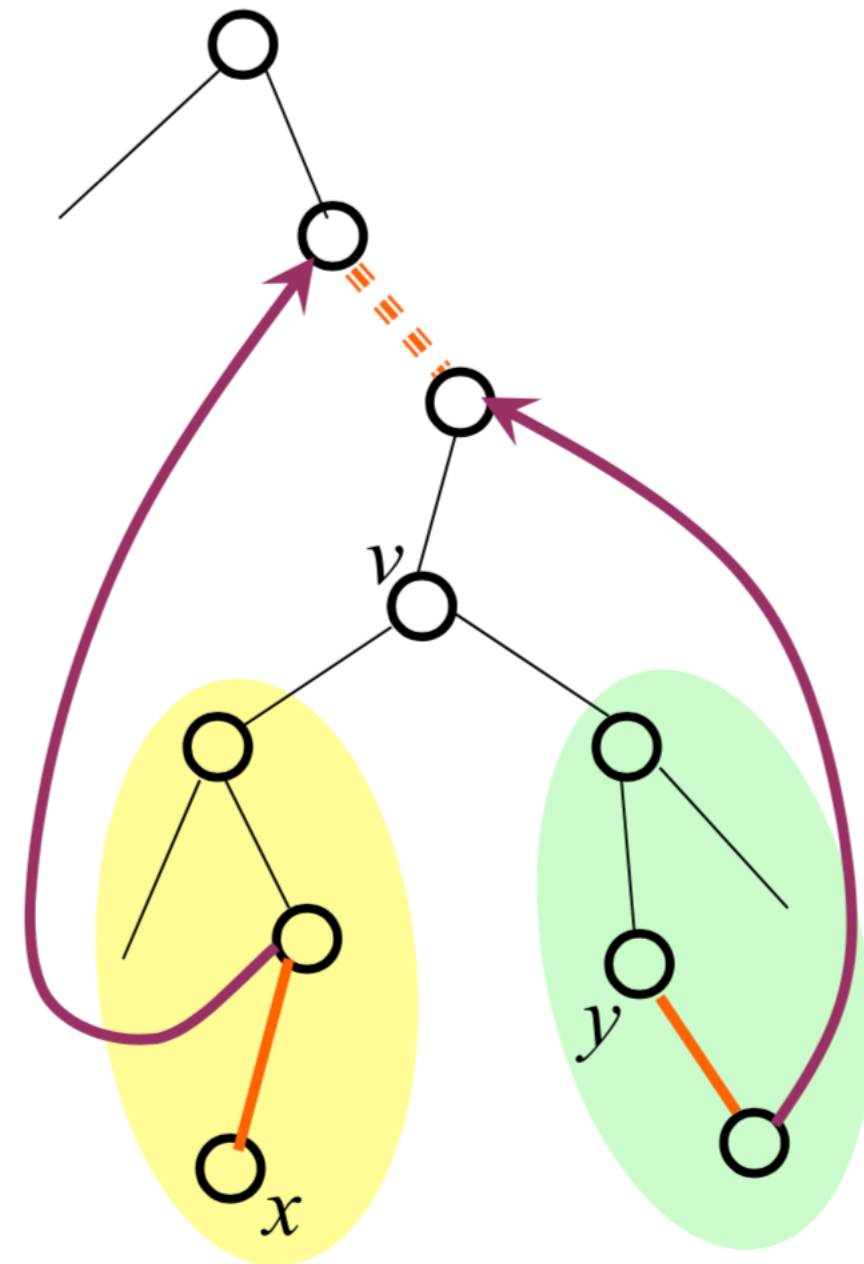
suppose that **every** subtree of v has a back edge to a proper ancestor of v , and, exactly one of x, y is a descendant of v .

Case 1.2: another is an ancestor of v



Case 2

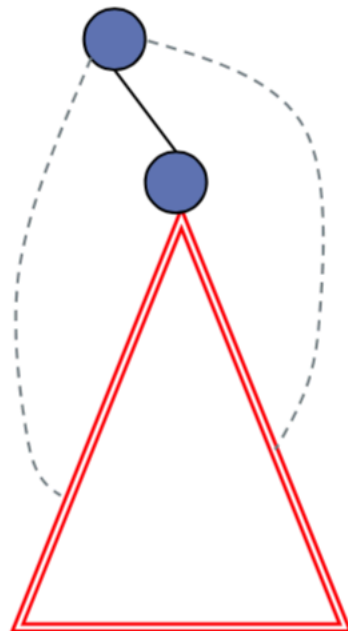
suppose that **every** subtree of v has a back edge to a proper ancestor of v , and, both x, y are descendants of v .



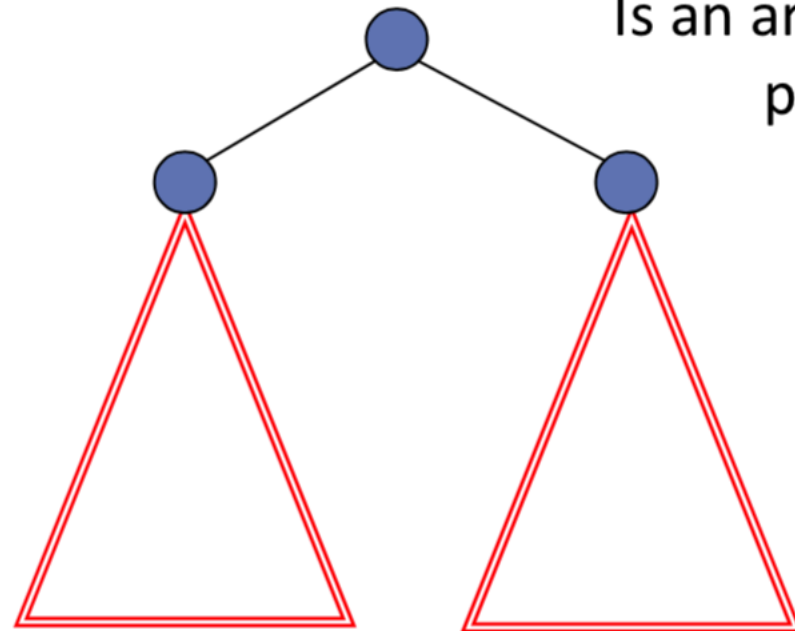
What about the root?

- One single DFS tree
 - We only consider each connected component
- Root AP \equiv Two or more sub-tree
 - The root is an articulation point

Not an
articulation point

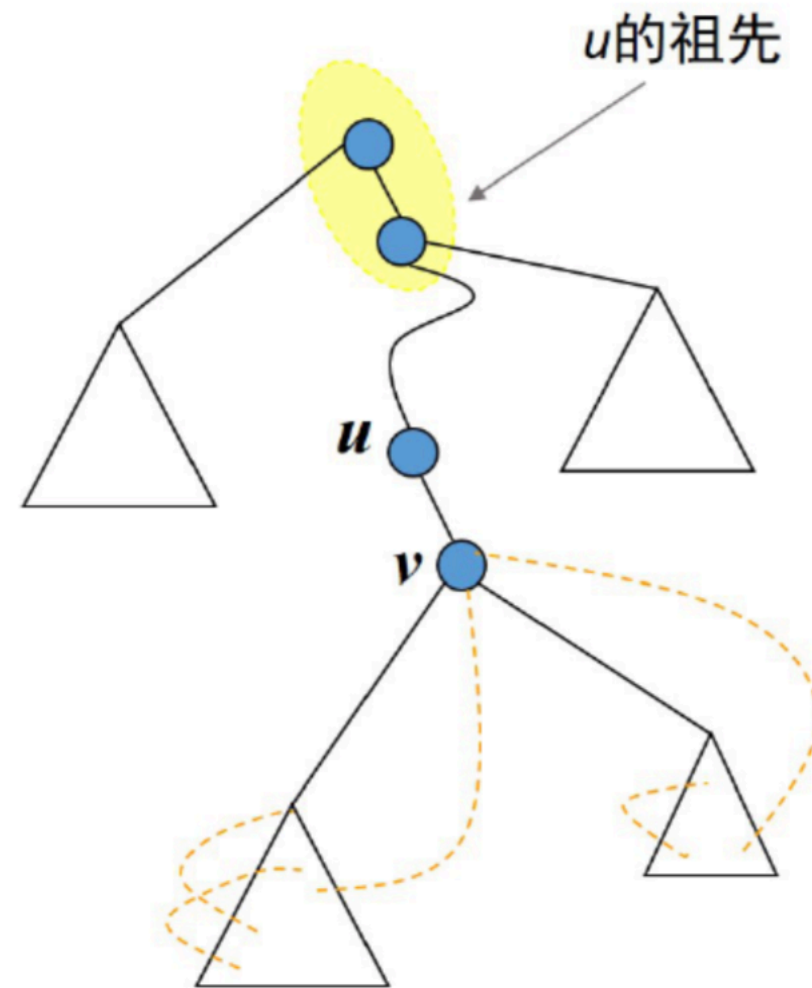


Is an articulation
point



Defining the Bridge

- Short definition
 - Removing uv leading to disconnection
- Long definition
 - Edge uv is a bridge iff node u and v are connected only by uv
- DFS Definition
 - Edge uv is a tree edge in DFS
 - There is no subtree rooted at v to any proper ancestor of v (including u)



Bridge Algorithm

Algorithm 11: BRIDGE-DFS(u)

```
1  $u.color := \text{GRAY}$  ;
2  $time := time + 1$  ;
3  $u.discoverTime := time$  ;
4  $u.back := u.discoverTime$  ;
5 foreach neighbor  $v$  of  $u$  do
6   if  $v.color = \text{WHITE}$  then
7     BRIDGE-DFS( $v$ ) ;
8      $u.back := \min\{u.back, v.back\}$  ;
9     if  $v.back > u.discoverTime$  then
10      Output  $uv$  as a bridge ;
11   else
12     if  $uv$  is BE then                                     /*  $v$  是  $u$  非父节点的祖先节点 */
13       $u.back := \min\{u.back, v.discoverTime\}$  ;
```

Other Traversal Problems

- **Orientation of an undirected graph**
 - Give each edge a direction
 - Satisfying pre-specified constraints
 - E.g., the “in-degree of each vertex is at least 1”
- **Possible or not?**
 - If possible, how to?
- **As for “in-degree ≥ 1 ”**
 - Orientation possible iff. the graph has at least a circle
 - Find the end point of some back edge
 - A second DFS from this end point

Other Traversal Problems

MST: Minimum Spanning Tree

- **Get MST in $O(m+n)$ time**
 - Given that edges weights are only 1 and 2
- **Graph traversal is sufficient**
 - DFS over “weight 1 edges” only
 - DFS over “weight 2 edges” only

Thank you!

Q & A