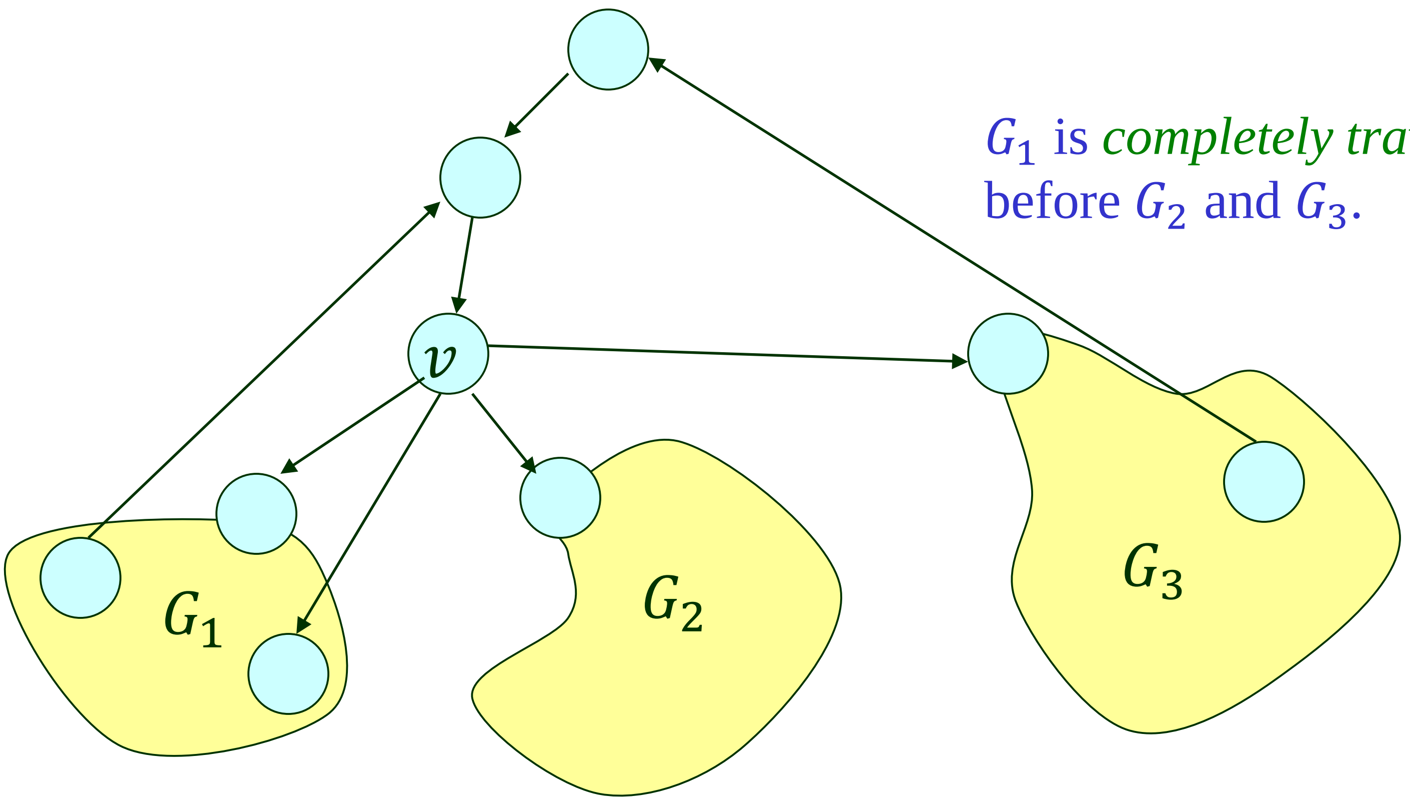
Depth-First Search

Idea: Keep going forward as long as there are unseen nodes to be visited. Backtrack when stuck.



G_1 is *completely traversed* before G_2 and G_3.

Color Map & Predecessor

Just like in BFS:

- ♦ color(v) == **green**: v is undiscovered and unprocessed
- ♦ color(v) == **red**: v has been discovered but not processed
- ♦ color(v) == **black**: v has been discovered and processed

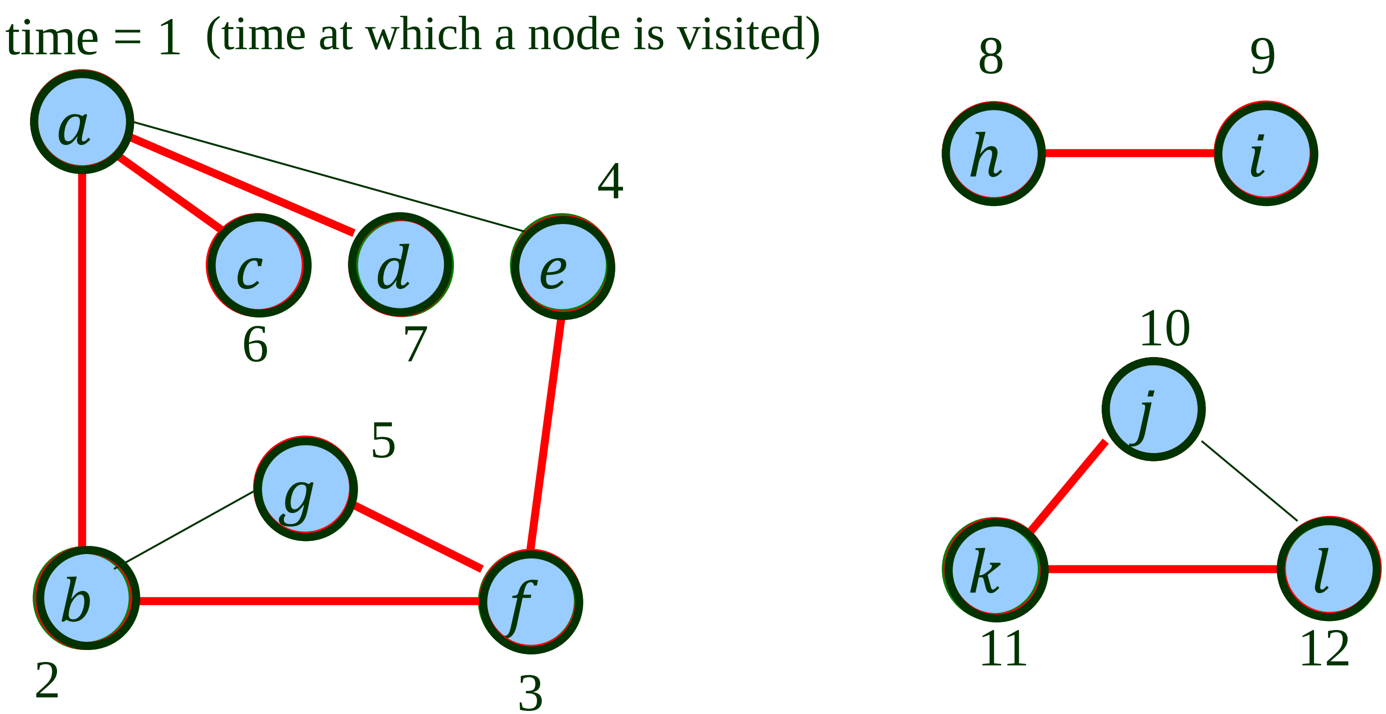
pred(v): predecessor of v in the search.

The DFS Algorithm

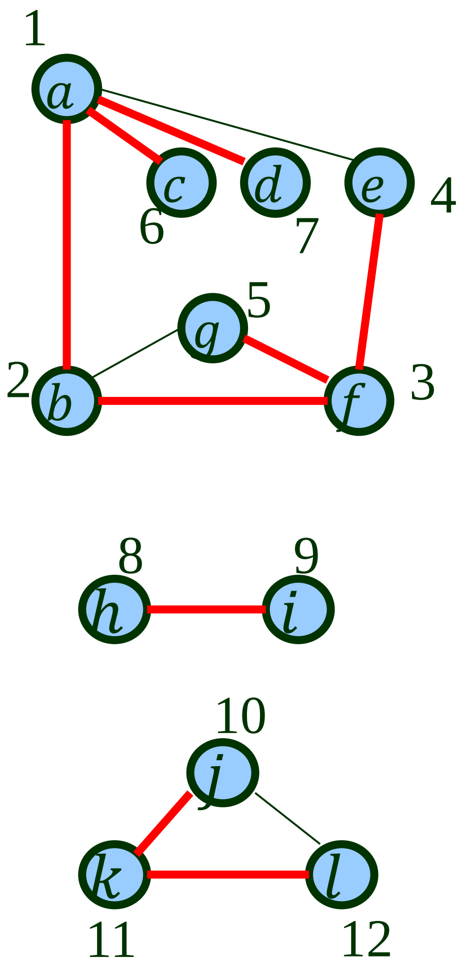
```
dfs(G):

    foreach node v in G
            color(v) = green
            pred(v) = null

    foreach node v in G
            if color(v) == green
                dfsVisit(G, v, color, pred)

    return pred
```

```
dfsVisit(G, u, color, pred):

    color(u) = red

    foreach neighbor v of u
            if color(v) == green
                pred(v) = u
                dfsVisit(G, v, color, pred)

    color(u) = black
```

A DFS Example



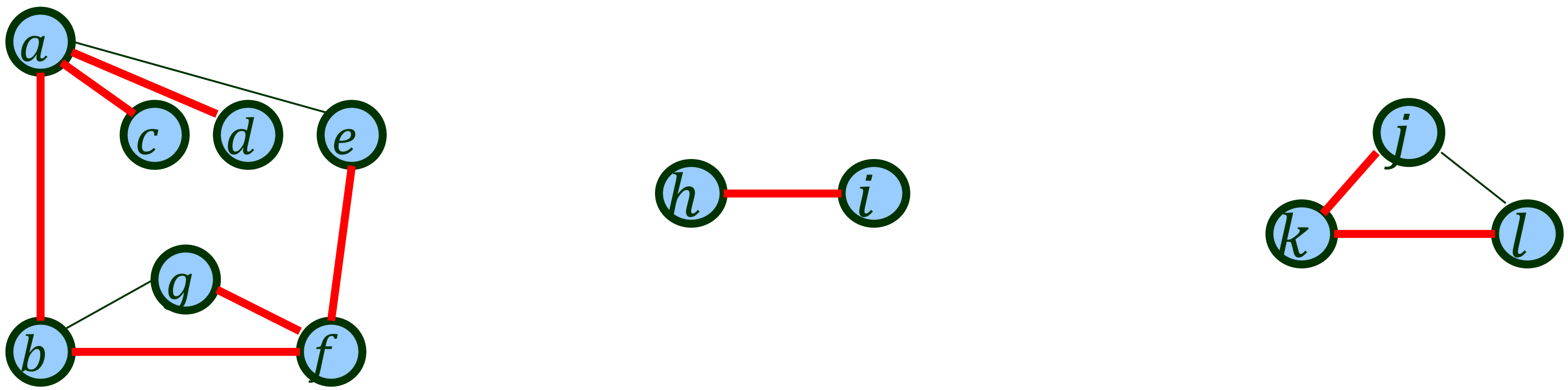
Recursive DFS Calls



dfs(G)
  dfsVisit(a)
    dfsVisit(b)
      dfsVisit(f)
        dfsVisit(e)
        dfsVisit(g)
    dfsVisit(c)
    dfsVisit(d)
  dfsVisit(h)
    dfsVisit(i)
  dfsVisit(j)
    dfsVisit(k)
      dfsVisit(l)

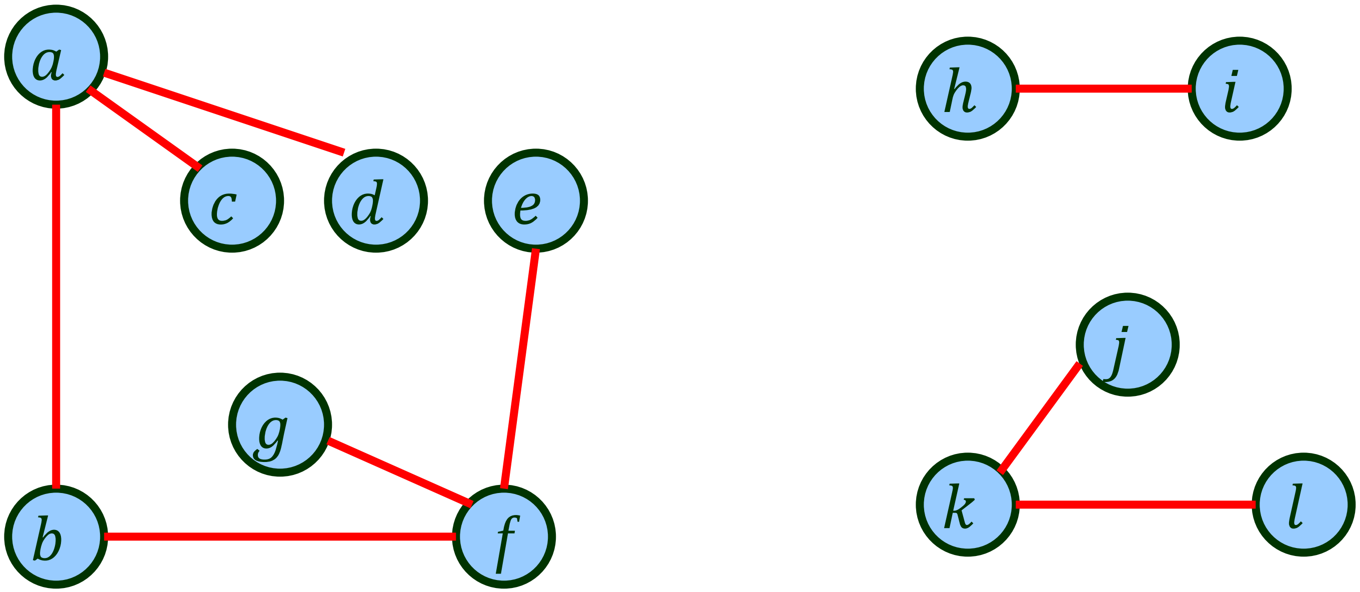
dfsVisit(v) explores ***every unvisited*** vertex reachable from v before it returns.

Predecessor Table



v	a	b	c	d	e	f	g	h	i	j	k	l
pred(v)	null	a	a	a	f	b	f	null	h	null	j	k

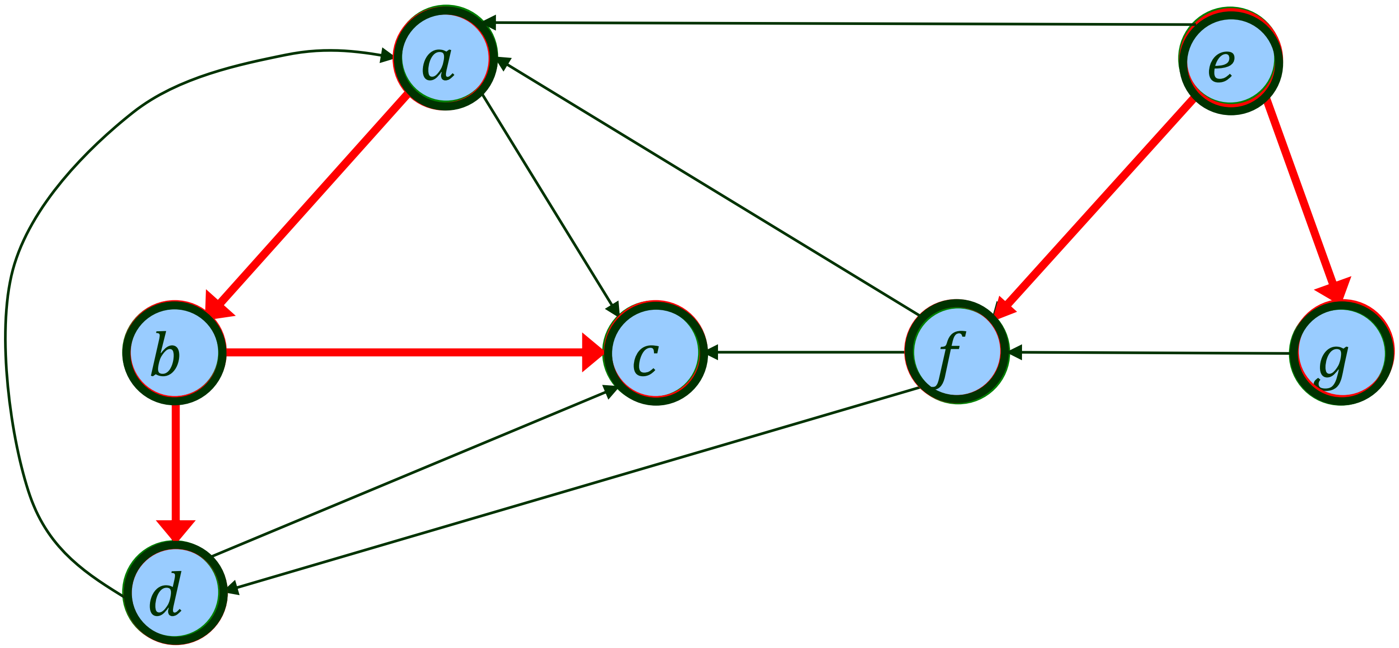
Depth-First Search Forest



Edges that, during DFS, lead to an unexplored vertex form a ***depth-first search forest.***

The DFS forest can be constructed from the predecessor table.

DFS on a Directed Graph



The Green-Path Theorem

- color(v) == **green**: v is undiscovered and unprocessed

DFS forest of a (directed or undirected) graph G:

Node v is a descendant of a node u *if and only if* at the time that the search discovers u, there is a path from u to v consisting *entirely* of **green** nodes.

v will be discovered after u
and before DFS backtracks to u.

$\Updownarrow$

v is in the DFS subtree rooted at u.

Running Time of DFS

$\boldsymbol{\Theta(|V| + |E|)}$ if we use an adjacency list or HashMap/HashSet.

- dfsVisit is called exactly once for each node.
  - $|V|$ such calls in total.
  - Each call colors the node **red** and then **black**, which takes $O(1)$ time.

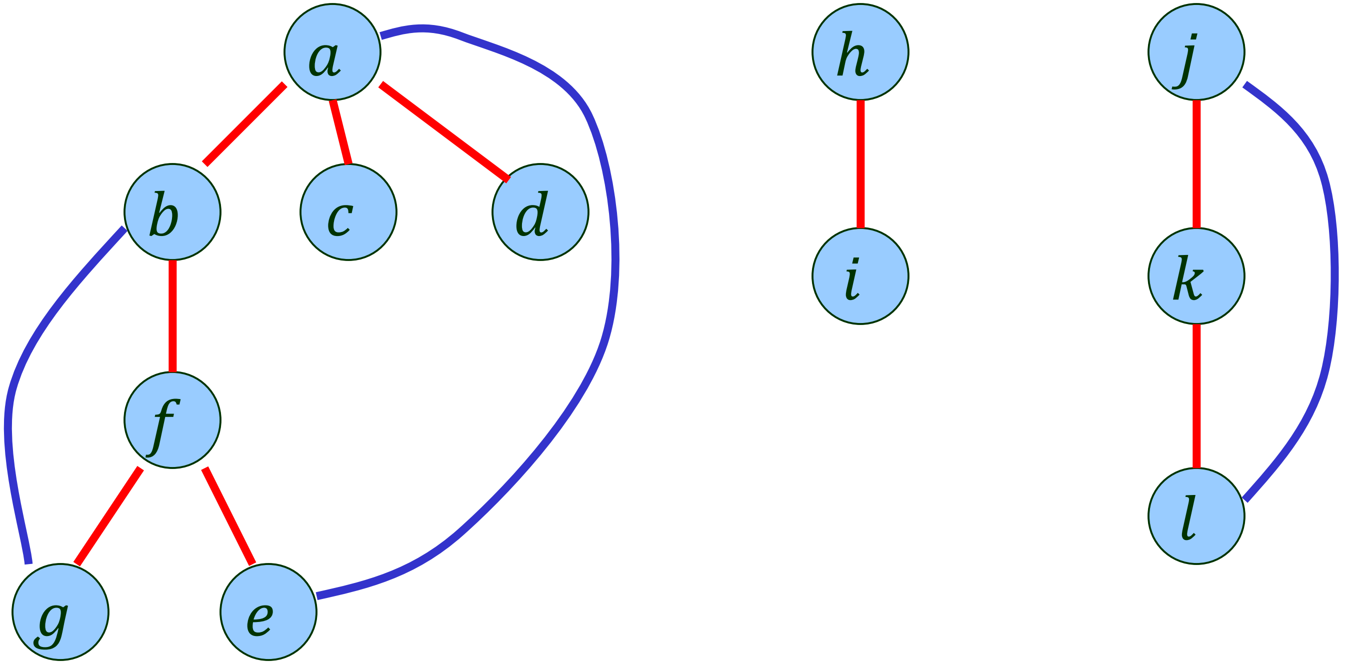
- Each edge is examined $O(1)$ time.

$\boldsymbol{\Theta(|V|^2)}$ if we use an adjacency matrix.

Edge Classification – Undirected Graphs

1. **Tree edges** are those in the DFS forest.

2. **Back edges** go from a vertex to one of its ancestors.

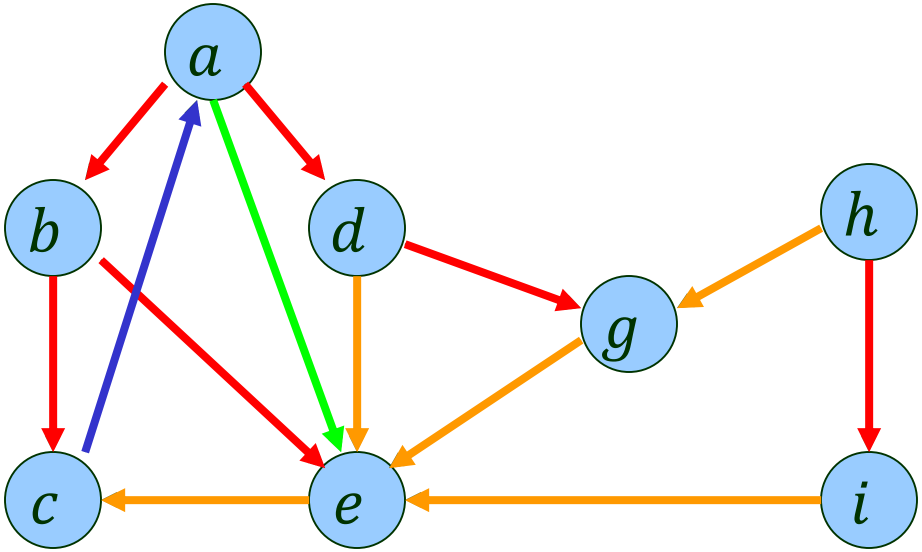


This DFS forest consists of three DFS trees.

Edge Classification – Directed Graphs

Besides **tree edges** and **back edges**, there are also

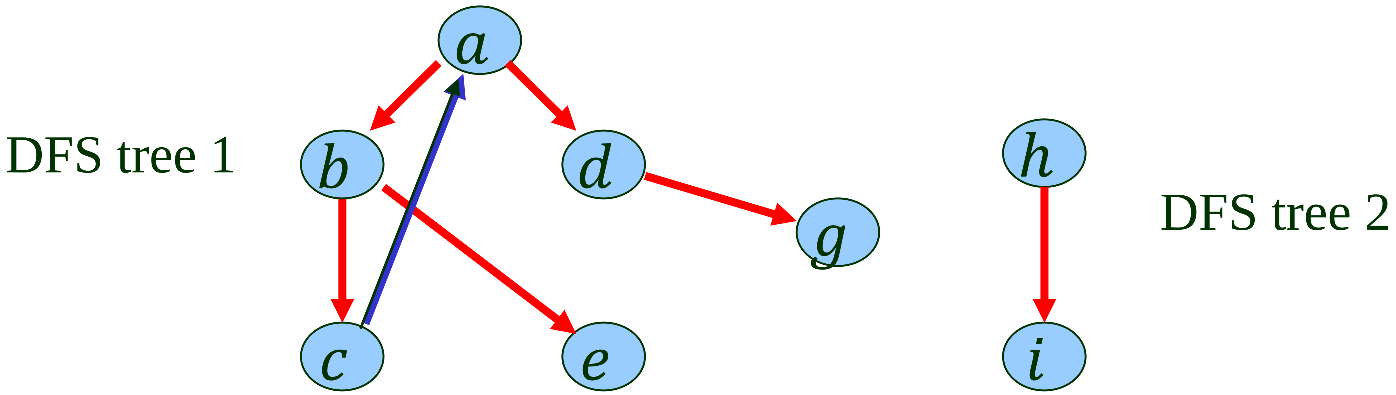
3. **Forward edges** go from a vertex to one of its descendants.
4. **Cross edges**: all other edges.



If all the edges were undirected, forward and cross edges would be either tree or back edges.

How to Tell Them Apart?

Tree edges lead to **green** (new) nodes and form DFS trees.



Add every other *separately* to the DFS forest.

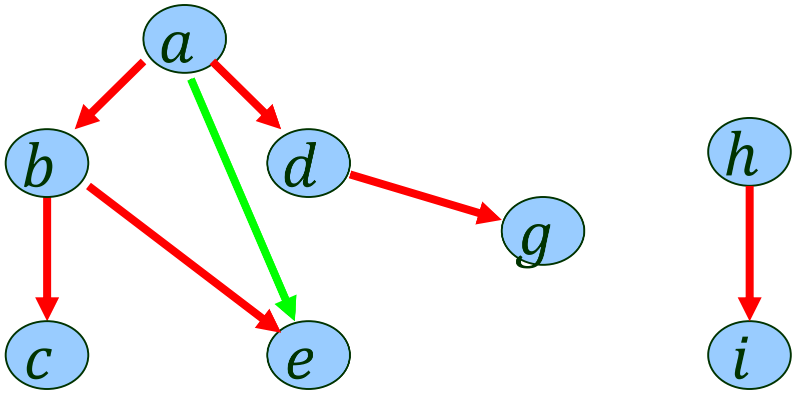
Back edges connect nodes in the *same DFS tree*.

- **back edge**: descendent → ancestor

Forward Edge

Forward edges also connect nodes in the *same DFS tree*.

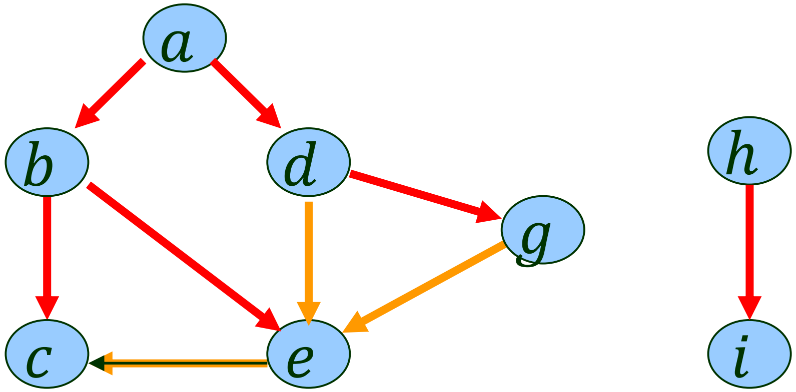
- **forward edge**: ancestor → descendent



Cross Edge

Each **cross edge** is one of two cases:

- Its two vertices are in the same DFS tree but ***not*** related.



- They are in ***different*** DSF trees.