Specialized DFS

cycles, bridges, articulation points

beOI Training
(slides by François Aubry)

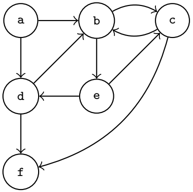
OLYMPIADE BELGE D'INFORMATIQUE
BELGISCHE INFORMATICA-OLYMPIADE

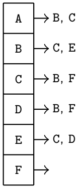
October 8, 2016

Table of Contents

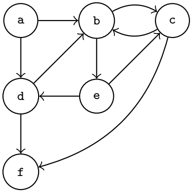
```
enum state {UNVISITED, OPENED, CLOSED};
vector<int> adj[N];
state st[N];

void dfs(int u) {
    st[v] = OPENED;
    for (int v : adj[u])
        if (st[u] == UNVISITED)
            dfs(v);
    st[v] = CLOSED;
}

// in main()
fill(st, st+n, UNVISITED);
for (int u = 0; u < n; i++)
    if (st[u] == UNVISITED)
        dfs(u);
```

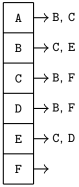
An extended version of DFS: remember if node is finished or not



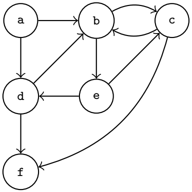


DFS execution from node a



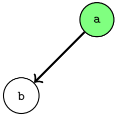


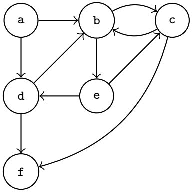
DFS execution from node a



A	→ B, C
B	→ C, E
C	→ B, F
D	→ B, F
E	→ C, D
F	→

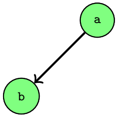
DFS execution from node a

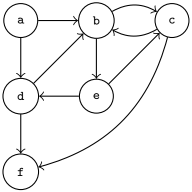




A	→ B, C
B	→ C, E
C	→ B, F
D	→ B, F
E	→ C, D
F	→

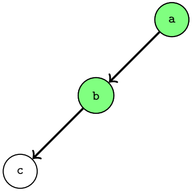
DFS execution from node a

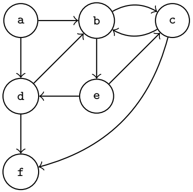




A	→ B, C
B	→ C, E
C	→ B, F
D	→ B, F
E	→ C, D
F	→

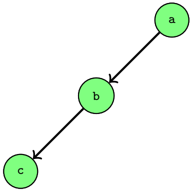
DFS execution from node a

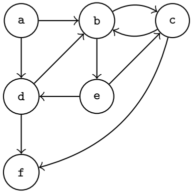


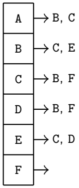


A	→ B, C
B	→ C, E
C	→ B, F
D	→ B, F
E	→ C, D
F	→

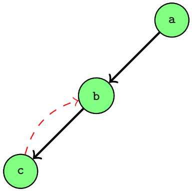
DFS execution from node a

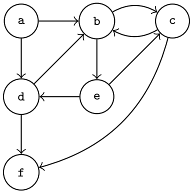






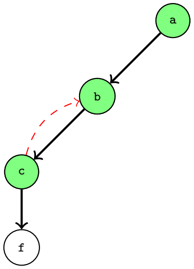
DFS execution from node a

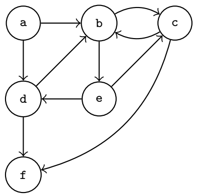


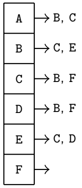


A	→ B, C
B	→ C, E
C	→ B, F
D	→ B, F
E	→ C, D
F	→

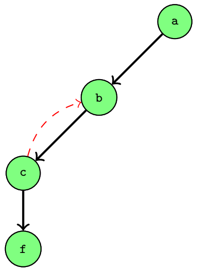
DFS execution from node a

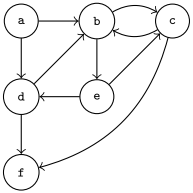






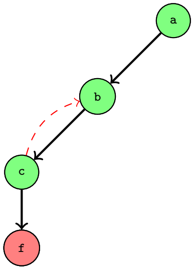
DFS execution from node a

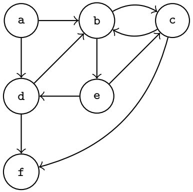




A	→ B, C
B	→ C, E
C	→ B, F
D	→ B, F
E	→ C, D
F	→

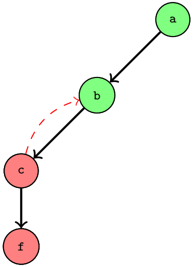
DFS execution from node a

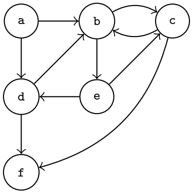




A	→ B, C
B	→ C, E
C	→ B, F
D	→ B, F
E	→ C, D
F	→

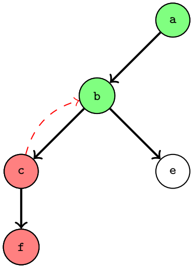
DFS execution from node a

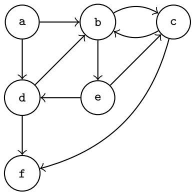




A	→ B, C
B	→ C, E
C	→ B, F
D	→ B, F
E	→ C, D
F	→

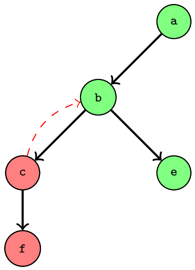
DFS execution from node a

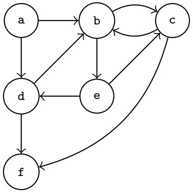


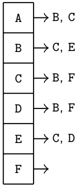


A	→ B, C
B	→ C, E
C	→ B, F
D	→ B, F
E	→ C, D
F	→

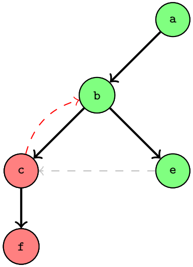
DFS execution from node a

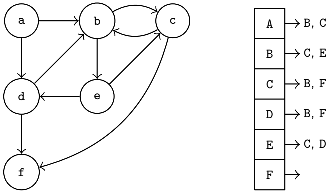




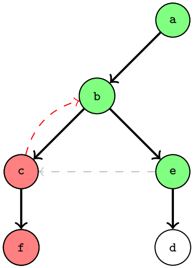


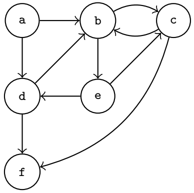
DFS execution from node a





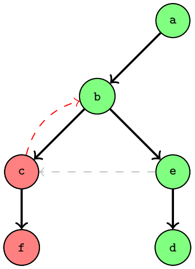
DFS execution from node a

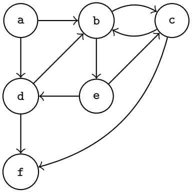




A	B, C
B	C, E
C	B, F
D	B, F
E	C, D
F	

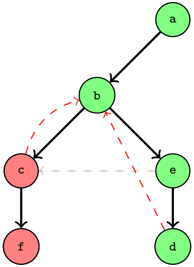
DFS execution from node a

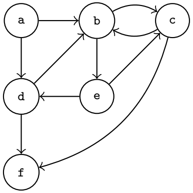




A	→ B, C
B	→ C, E
C	→ B, F
D	→ B, F
E	→ C, D
F	→

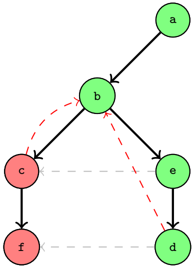
DFS execution from node a

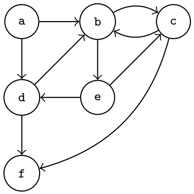


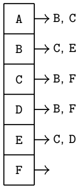


A	→ B, C
B	→ C, E
C	→ B, F
D	→ B, F
E	→ C, D
F	→

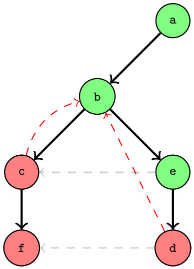
DFS execution from node a

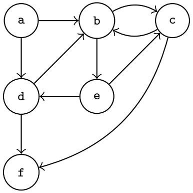


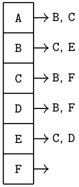




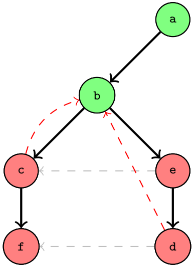
DFS execution from node a

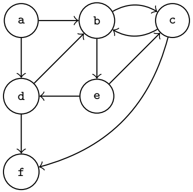






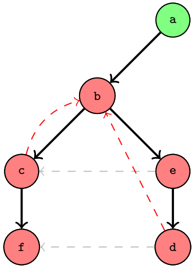
DFS execution from node a

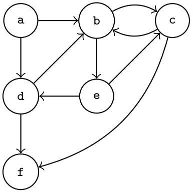


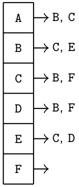


A	→ B, C
B	→ C, E
C	→ B, F
D	→ B, F
E	→ C, D
F	→

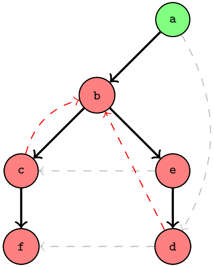
DFS execution from node a

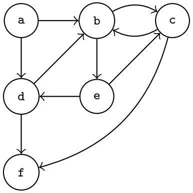


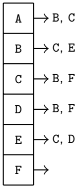




DFS execution from node a







DFS execution from node a

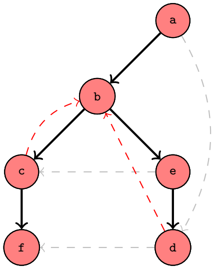


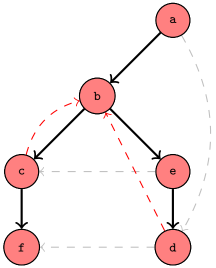
Table of Contents

Let's start with **cycles**.

How can we detect with DFS whether a graph is acyclic or not?

Let's start with **cycles**.

How can we detect with DFS whether a graph is acyclic or not?

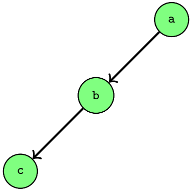


The red edges belong to cycles, they are called **back edges**.

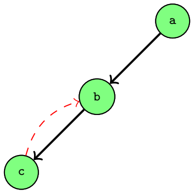
A graph is acyclic if and only if DFS does not yield back edges.

How to distiguish these edges from the others?

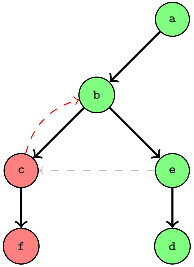
How to distiguish these edges from the others?



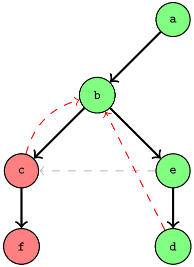
How to distiguish these edges from the others?



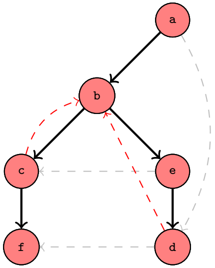
How to distiguish these edges from the others?



How to distiguish these edges from the others?



How to distiguish these edges from the others?



They are edges from a green (OPENED) node to another green (OPENED) node!

To implement this we simply add a check while listing neighbours.

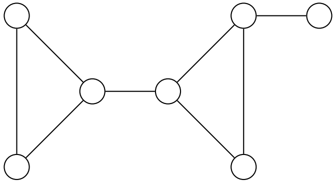
Checks if node *u* (which is OPENED) points to another OPENED node.

```
bool hasCycle = false;

void dfs(int u) {
    st[v] = OPENED;
    for (int v : adj[u]) {
        if (st[u] == UNVISITED)
            dfs(v);
        else if (st[u] == OPENED)
            hasCycle = true;
    }
    st[v] = CLOSED;
}
```

Table of Contents

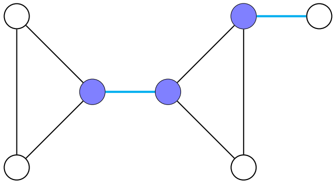
An **articulation point** in an undirected graph is a node such that its removal disconnects the graph.

A **bridge** in an undirected graph is an edge such that its removal disconnects the graph.



An **articulation point** in an undirected graph is a node such that its removal disconnects the graph.

A **bridge** in an undirected graph is an edge such that its removal disconnects the graph.



— bridges

● articulation points

How would you compute this?

How would you compute this?

Naive algorithm (for bridges):

For every edge (x, y), remove it and check with BFS or DFS whether x and y remain connected.

Complexity: $O(E \cdot (V + E)) = O(E \cdot V + E^2)$ TLE in big graphs.

We will solve it in **linear time** with a single DFS.

Observation:

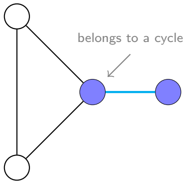
Bridges can never belong to cycles.

Is this true for articulation points?

Observation:

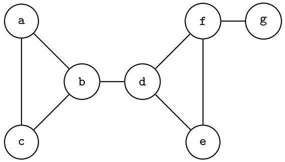
Bridges can never belong to cycles.

Is this true for articulation points? **NO**.



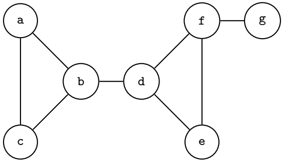
So we essentially need to find which edges belong to cycles.

We already say that DFS allows to find cycles.



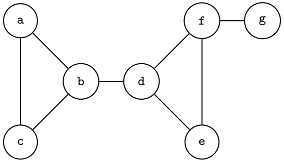
A	→ B, C
B	→ A, C, D
C	→ A, B
D	→ B, E, F
E	→ D, F
F	→ D, E, G
G	→ F

DFS execution from node a



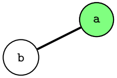
A	$\rightarrow$ B, C
B	$\rightarrow$ A, C, D
C	$\rightarrow$ A, B
D	$\rightarrow$ B, E, F
E	$\rightarrow$ D, F
F	$\rightarrow$ D, E, G
G	$\rightarrow$ F

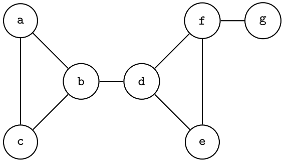
DFS execution from node a



A	B, C
B	A, C, D
C	A, B
D	B, E, F
E	D, F
F	D, E, G
G	F

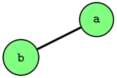
DFS execution from node a

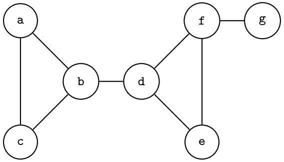


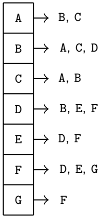


A	$\rightarrow$ B, C
B	$\rightarrow$ A, C, D
C	$\rightarrow$ A, B
D	$\rightarrow$ B, E, F
E	$\rightarrow$ D, F
F	$\rightarrow$ D, E, G
G	$\rightarrow$ F

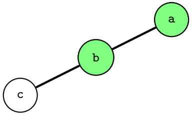
DFS execution from node a

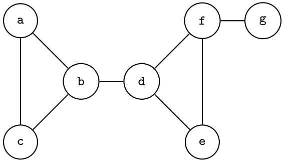






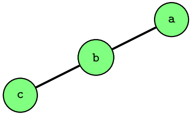
DFS execution from node a

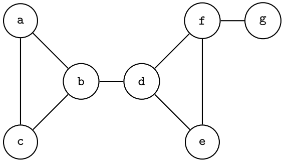




A	→ B, C
B	→ A, C, D
C	→ A, B
D	→ B, E, F
E	→ D, F
F	→ D, E, G
G	→ F

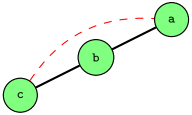
DFS execution from node a

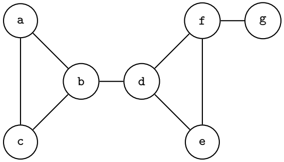




A	B, C
B	A, C, D
C	A, B
D	B, E, F
E	D, F
F	D, E, G
G	F

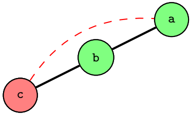
DFS execution from node a

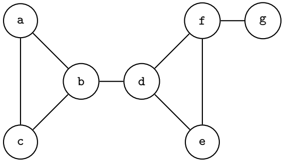




A	→ B, C
B	→ A, C, D
C	→ A, B
D	→ B, E, F
E	→ D, F
F	→ D, E, G
G	→ F

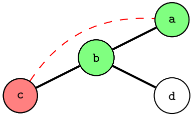
DFS execution from node a

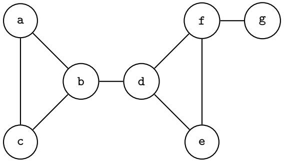


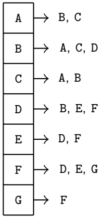


A	B, C
B	A, C, D
C	A, B
D	B, E, F
E	D, F
F	D, E, G
G	F

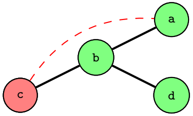
DFS execution from node a

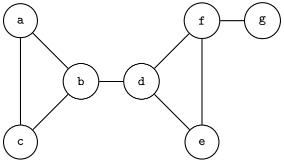


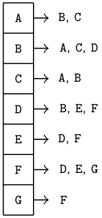




DFS execution from node a

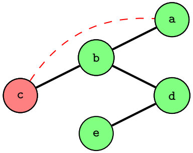


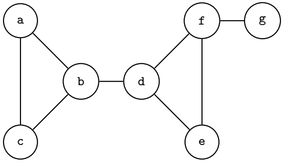




A	→ B, C
B	→ A, C, D
C	→ A, B
D	→ B, E, F
E	→ D, F
F	→ D, E, G
G	→ F

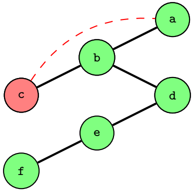
DFS execution from node a

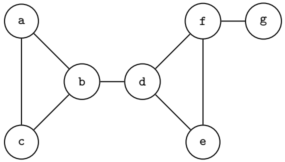




A	→ B, C
B	→ A, C, D
C	→ A, B
D	→ B, E, F
E	→ D, F
F	→ D, E, G
G	→ F

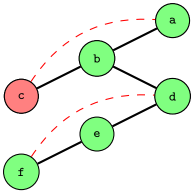
DFS execution from node a

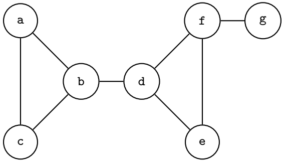




A	→ B, C
B	→ A, C, D
C	→ A, B
D	→ B, E, F
E	→ D, F
F	→ D, E, G
G	→ F

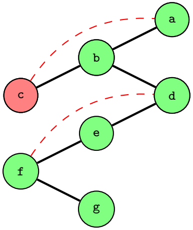
DFS execution from node a

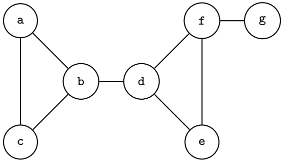




A	→ B, C
B	→ A, C, D
C	→ A, B
D	→ B, E, F
E	→ D, F
F	→ D, E, G
G	→ F

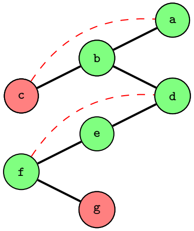
DFS execution from node a

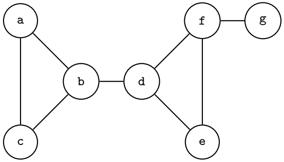




A	$\to$ B, C
B	$\to$ A, C, D
C	$\to$ A, B
D	$\to$ B, E, F
E	$\to$ D, F
F	$\to$ D, E, G
G	$\to$ F

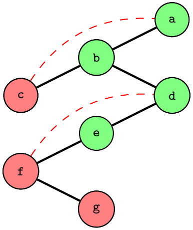
DFS execution from node a

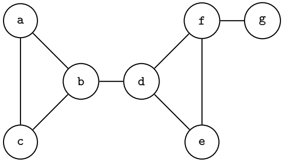


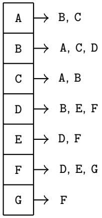


A	B, C
B	A, C, D
C	A, B
D	B, E, F
E	D, F
F	D, E, G
G	F

DFS execution from node a

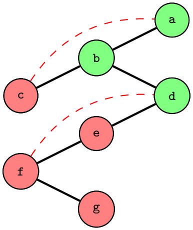


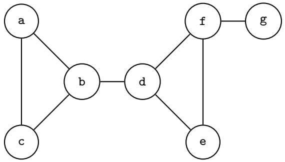




A	B, C
B	A, C, D
C	A, B
D	B, E, F
E	D, F
F	D, E, G
G	F

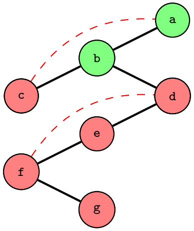
DFS execution from node a

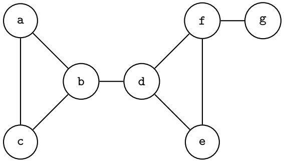


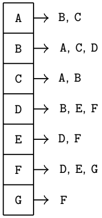


A	$\rightarrow$ B, C
B	$\rightarrow$ A, C, D
C	$\rightarrow$ A, B
D	$\rightarrow$ B, E, F
E	$\rightarrow$ D, F
F	$\rightarrow$ D, E, G
G	$\rightarrow$ F

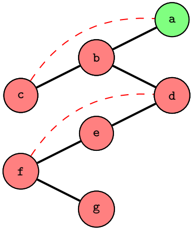
DFS execution from node a

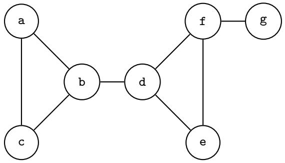






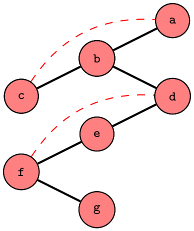
DFS execution from node a

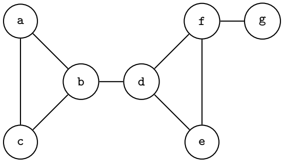




A	→ B, C
B	→ A, C, D
C	→ A, B
D	→ B, E, F
E	→ D, F
F	→ D, E, G
G	→ F

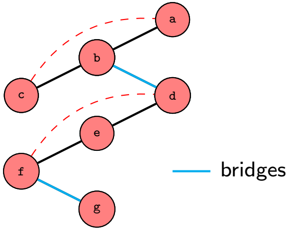
DFS execution from node a





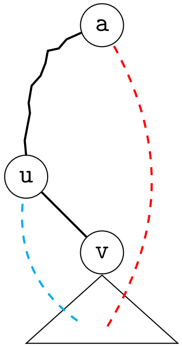
A	→ B, C
B	→ A, C, D
C	→ A, B
D	→ B, E, F
E	→ D, F
F	→ D, E, G
G	→ F

DFS execution from node a



We observe that an edge (u, v) is a bridge if and only if:

No node in the sub-tree of v has a link to u **or** one of its ancestors other than (u, v).



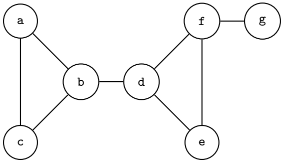
What should we add to our DFS to know this information?

We will **timestamp** the nodes as they are visited: *num*

Keep track of the node with minimum timestamp that we see: *low*

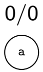
(u, v) is a bridge if and only if when we finish v, $low[v] > num[u]$

This is so because this means that in the sub-tree of v we did not see any ancestor of u.

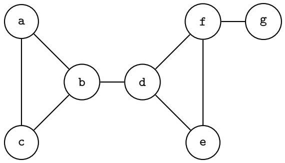


A	→ B, C
B	→ A, C, D
C	→ A, B
D	→ B, E, F
E	→ D, F
F	→ D, E, G
G	→ F

DFS execution from node a

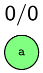


— bridges

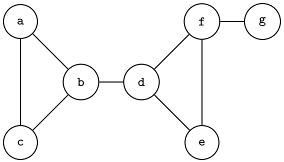


A	$\rightarrow$ B, C
B	$\rightarrow$ A, C, D
C	$\rightarrow$ A, B
D	$\rightarrow$ B, E, F
E	$\rightarrow$ D, F
F	$\rightarrow$ D, E, G
G	$\rightarrow$ F

DFS execution from node a

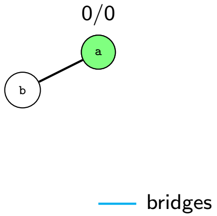


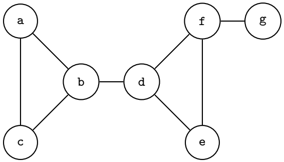
— bridges



A	B, C
B	A, C, D
C	A, B
D	B, E, F
E	D, F
F	D, E, G
G	F

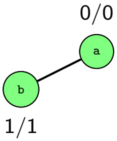
DFS execution from node a



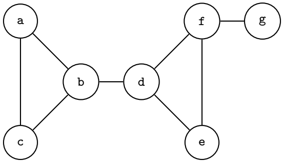


A	→ B, C
B	→ A, C, D
C	→ A, B
D	→ B, E, F
E	→ D, F
F	→ D, E, G
G	→ F

DFS execution from node a

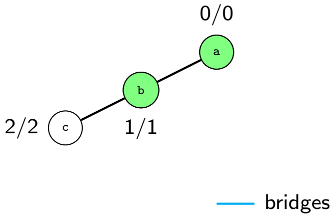


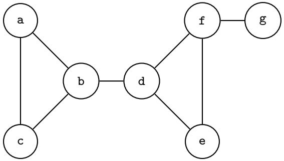
─ bridges



A	→ B, C
B	→ A, C, D
C	→ A, B
D	→ B, E, F
E	→ D, F
F	→ D, E, G
G	→ F

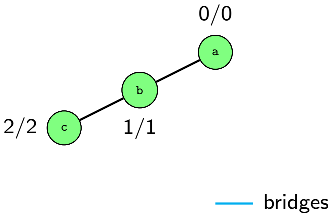
DFS execution from node a

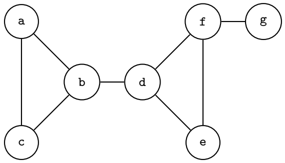




A	B, C
B	A, C, D
C	A, B
D	B, E, F
E	D, F
F	D, E, G
G	F

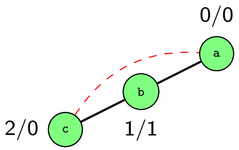
DFS execution from node a



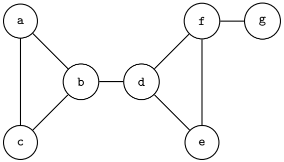


A	B, C
B	A, C, D
C	A, B
D	B, E, F
E	D, F
F	D, E, G
G	F

DFS execution from node a

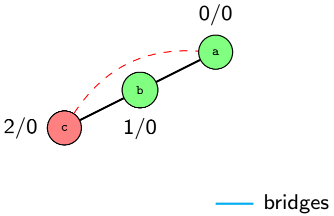


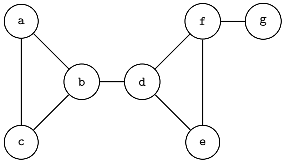
━ bridges



A	B, C
B	A, C, D
C	A, B
D	B, E, F
E	D, F
F	D, E, G
G	F

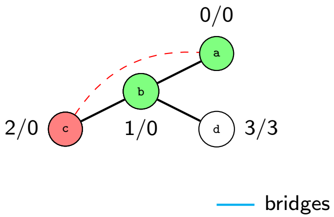
DFS execution from node a

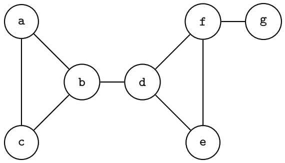




A	$\rightarrow$ B, C
B	$\rightarrow$ A, C, D
C	$\rightarrow$ A, B
D	$\rightarrow$ B, E, F
E	$\rightarrow$ D, F
F	$\rightarrow$ D, E, G
G	$\rightarrow$ F

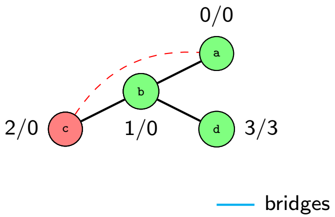
DFS execution from node a

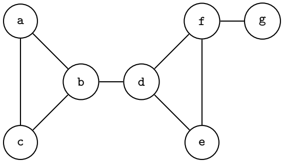




A	→ B, C
B	→ A, C, D
C	→ A, B
D	→ B, E, F
E	→ D, F
F	→ D, E, G
G	→ F

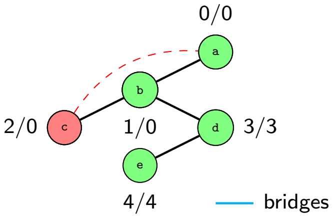
DFS execution from node a

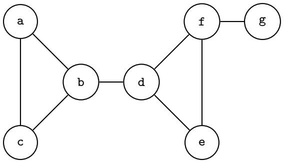




A	→ B, C
B	→ A, C, D
C	→ A, B
D	→ B, E, F
E	→ D, F
F	→ D, E, G
G	→ F

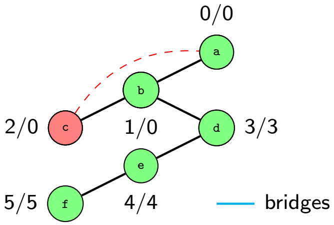
DFS execution from node a

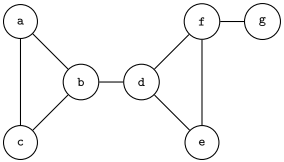


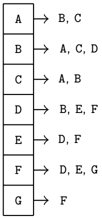


A	→ B, C
B	→ A, C, D
C	→ A, B
D	→ B, E, F
E	→ D, F
F	→ D, E, G
G	→ F

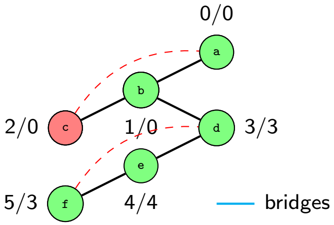
DFS execution from node a

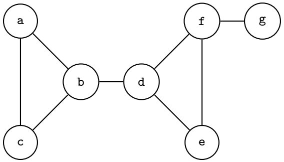






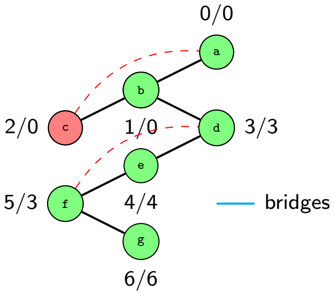
DFS execution from node a

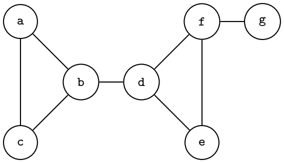




A	B, C
B	A, C, D
C	A, B
D	B, E, F
E	D, F
F	D, E, G
G	F

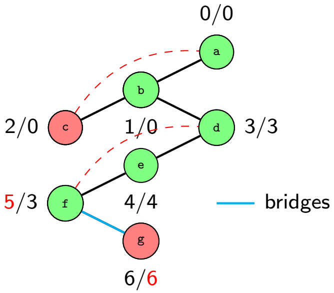
DFS execution from node a

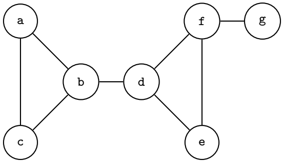


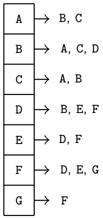


A	B, C
B	A, C, D
C	A, B
D	B, E, F
E	D, F
F	D, E, G
G	F

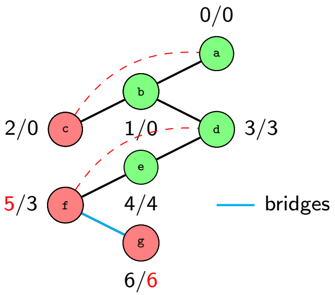
DFS execution from node a

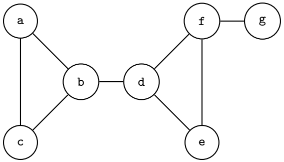






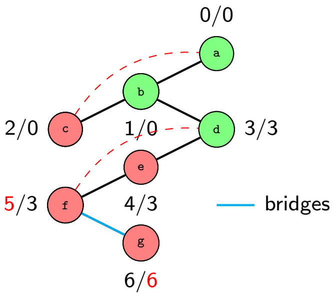
DFS execution from node a

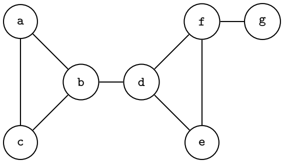




A	→ B, C
B	→ A, C, D
C	→ A, B
D	→ B, E, F
E	→ D, F
F	→ D, E, G
G	→ F

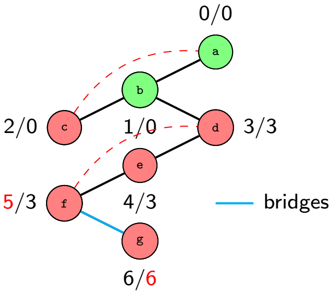
DFS execution from node a

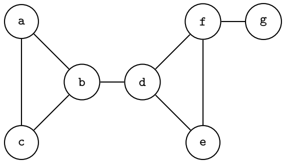




A	→ B, C
B	→ A, C, D
C	→ A, B
D	→ B, E, F
E	→ D, F
F	→ D, E, G
G	→ F

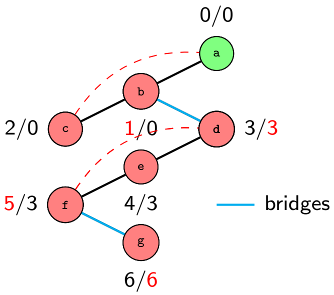
DFS execution from node a

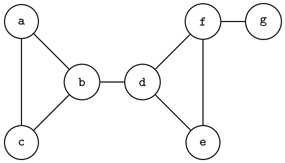




A	B, C
B	A, C, D
C	A, B
D	B, E, F
E	D, F
F	D, E, G
G	F

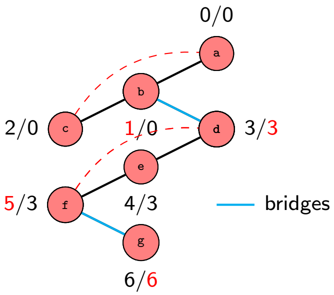
DFS execution from node a





A	→ B, C
B	→ A, C, D
C	→ A, B
D	→ B, E, F
E	→ D, F
F	→ D, E, G
G	→ F

DFS execution from node a

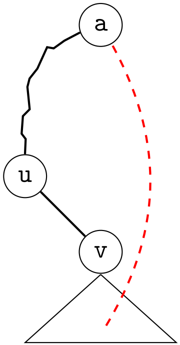


So what about **articulation points**?

So what about **articulation points**?

We observe that a node u is an articulation point if and only if:

No node in the sub-tree of v has a link to one ancestor of u other than (u, v).

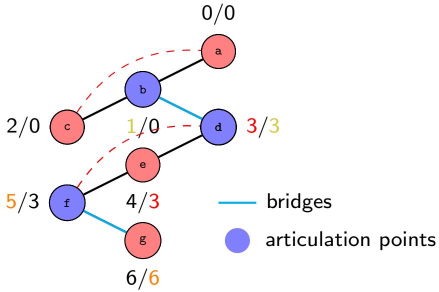


Note that a link to u does not help here.

What does this mean in terms of *num* and *low*?

What does this mean in terms of *num* and *low*?

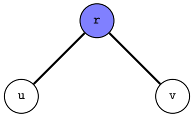
A non-root node u is an articulation point if and only if when we finish v, $low[v] \geq num[u]$



What about the root?

What about the root?

Root is articulation if and only if it has more than one child.



Removing node r disconnects the graph.

If there was another path from u to v, v would not be a child of r.

```
int num[N], low[N], root, rootChildren, cnt = 0;

void dfs(int u, int parent) {
    num[u] = low[u] = cnt++;
    vis[u] = true;
    for (int v : adj[u]) {
        if (!vis[u]) {
            dfs(v, u);
            if (u == root) rootChildren++;
            if (low[v] >= num[u] && u != root)
                // u is an articulation point
            if (low[v] > num[u])
                // (u,v) is a bridge
            low[u] = min(low[u], low[v]);
        } else if (v != parent)
            low[u] = min(low[u], num[v]);
    }
}

// loop in main():
if (!vis[u]) {
    root = u, rootChildren = 0;
    dfs(u, -1);
    if(rootChildren > 1)
        // u is an articulation point
}
```