NCKU Programming Contest Training Course
Strong Connected Component(SCC)
2018/04/25

Syuan-Yi Lin(petermouse)
petermouselin@gmail.com

Department of Computer Science and Information Engineering
National Cheng Kung University
Tainan, Taiwan

made by
tommy5198 & free999 &
electorn & petermouse

Outline

- Articulation/Bridge
  (in undirected graph)

- Strongly Connected Component(SCC)
  (in directed graph)

made by tommy5198
& petermouse

Graph

- connected graph/component

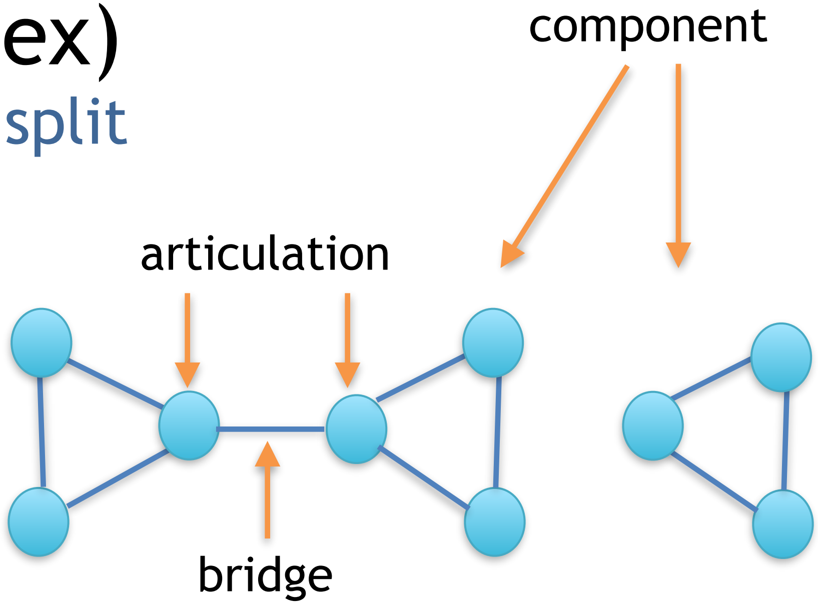
  connected graph/component iff all pairwise vertices exist at least one path & no more vertices can be added

- articulation(cut-vertex)

  remove articulation vertex split one component to two

- bridge(cut-edge)

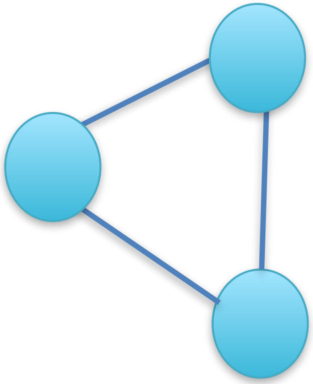
  same as articulation



made by tommy5198

Articulation/Bridge

- Find Articulation in Graph
  - Graph become non-connected if remove a Articulation. V times DFS = O(V*(V+E)) —> too slow!
  - Vertex is not Articulation if can find alternative path —> find cycle!
  - Use DFS —> O(V+E)



Articulation/Bridge

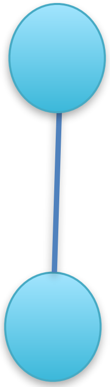
- Concept

  - DFS traversel will construct relationships in a tree
  - if vertex u's children can't back to u's ancestors

   —> u is Articulation

  - if vertex u is root and has at least 2 child

  —> u is Articulation

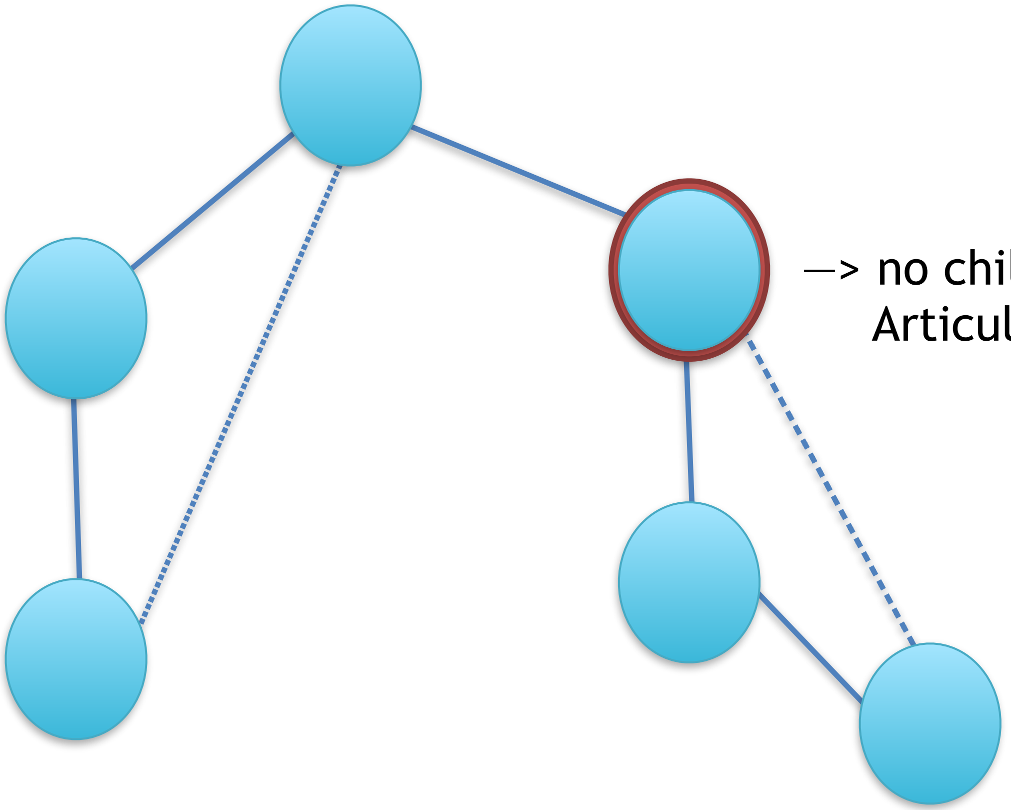


- Bridge?

  - two Articulation u, v have an edge —> (u, v) is Bridge!

made by tommy5198
& petermouse

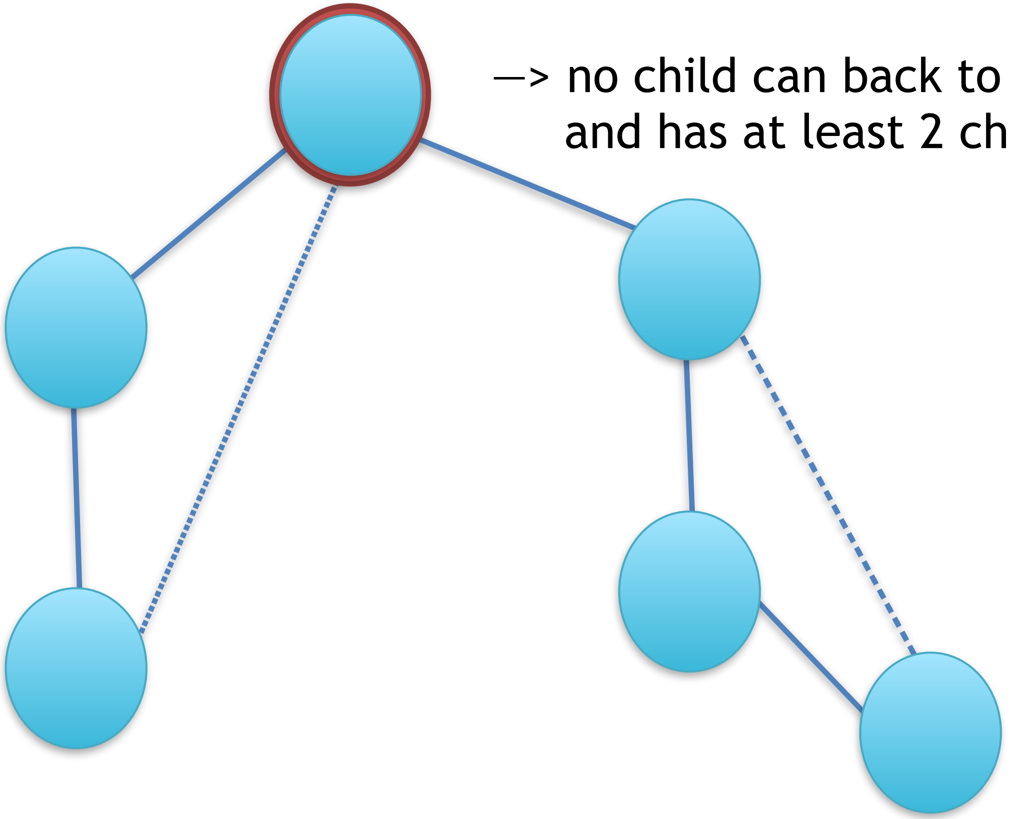
Articulation/Bridge



—> no child can back to ancestor.
Articulation!

made by tommy5198

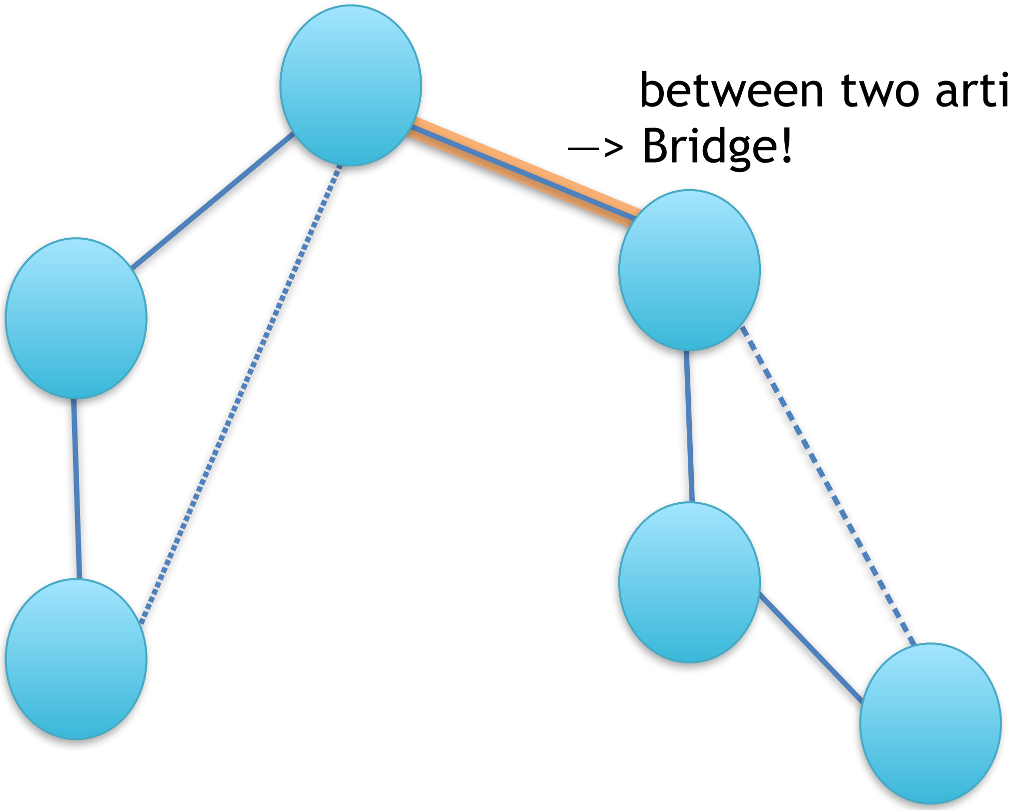
Articulation/Bridge



—> no child can back to ancestor
and has at least 2 child. Articulation!

made by tommy5198

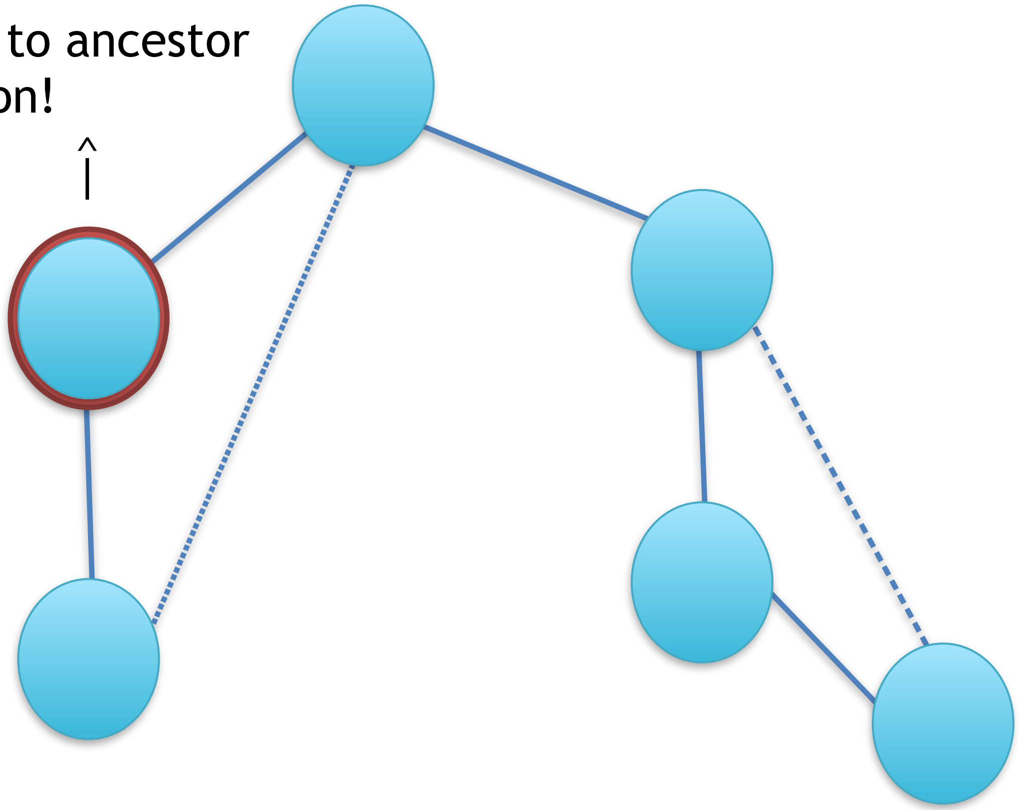
Articulation/Bridge



between two articulations
—> Bridge!

Articulation/Bridge

child can back to ancestor
NOT Articulation!



Articulation/Bridge

- dfn[u] = DFS traversal order
  - first visit time each vertex u in DFS

- low[u] = min(dfn[u], lowest low[v])
  - if edge (u,v) exist and v is not u's parent

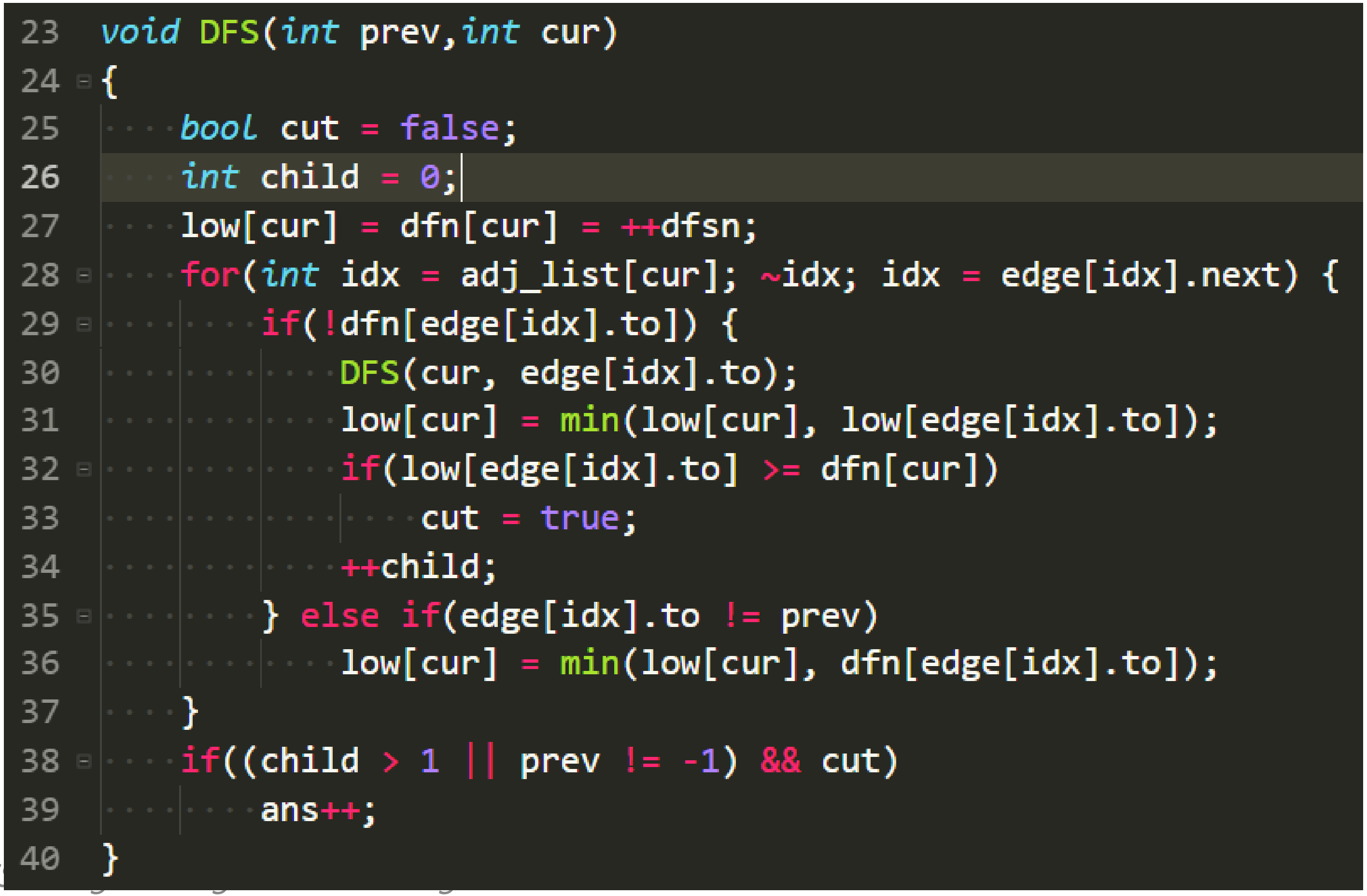
Articulation/Bridge

- Articulation
  - if vertex u’s children can’t back to u’s ancestors
  —> dfn[u] <= low[v], v is u’s child

  - if vertex u is root and has at least 2 child
  —> count child >= 2
- Bridge?
  - two Articulation u, v —> dfn[u] < low[v], v is u’s child

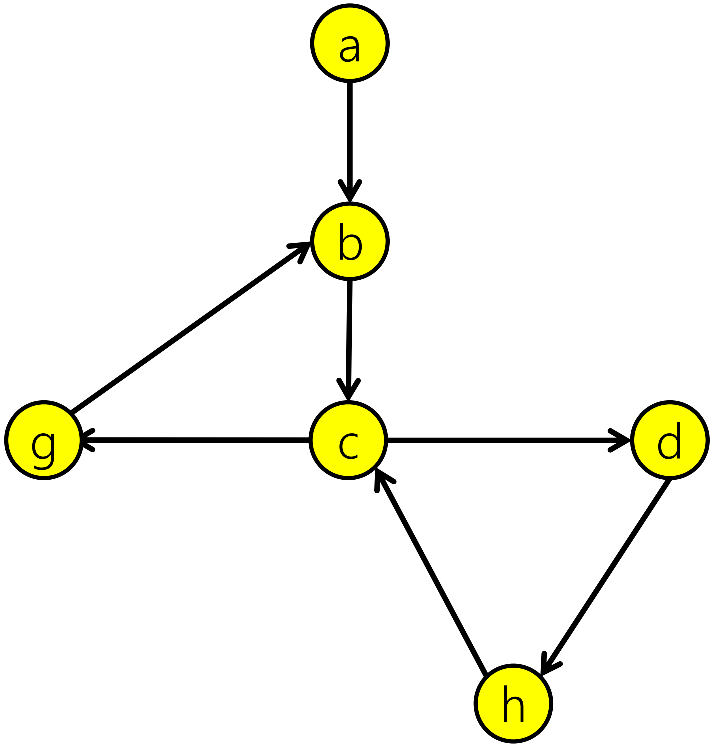
Cut vertex

- code



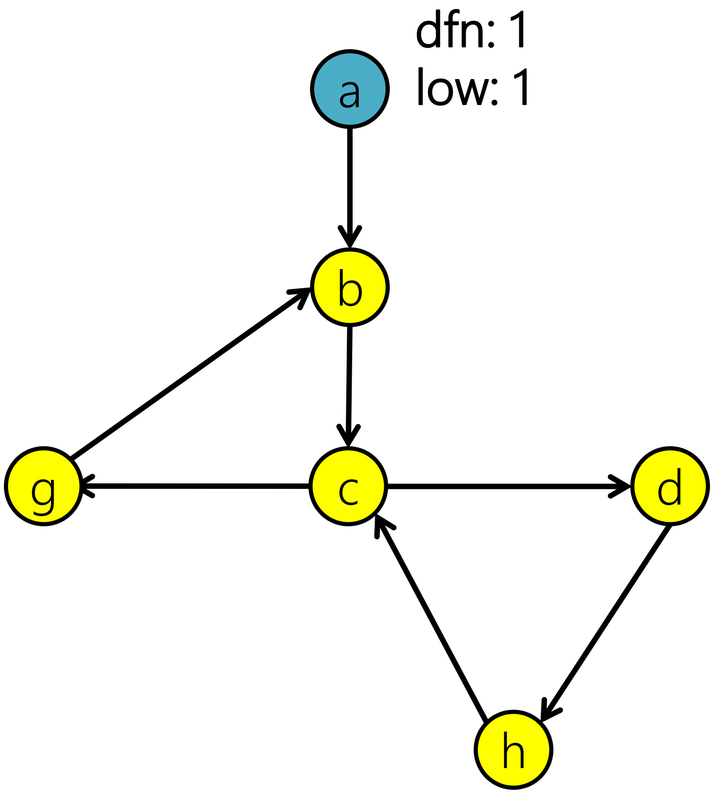
```cpp
void DFS(int prev,int cur)
{
    bool cut = false;
    int child = 0;
    low[cur] = dfn[cur] = ++dfsn;
    for(int idx = adj_list[cur]; ~idx; idx = edge[idx].next) {
        if(!dfn[edge[idx].to]) {
            DFS(cur, edge[idx].to);
            low[cur] = min(low[cur], low[edge[idx].to]);
            if(low[edge[idx].to] >= dfn[cur])
                cut = true;
            ++child;
        } else if(edge[idx].to != prev)
            low[cur] = min(low[cur], dfn[edge[idx].to]);
    }
    if((child > 1 || prev != -1) && cut)
        ans++;
}
```

Cut vertex

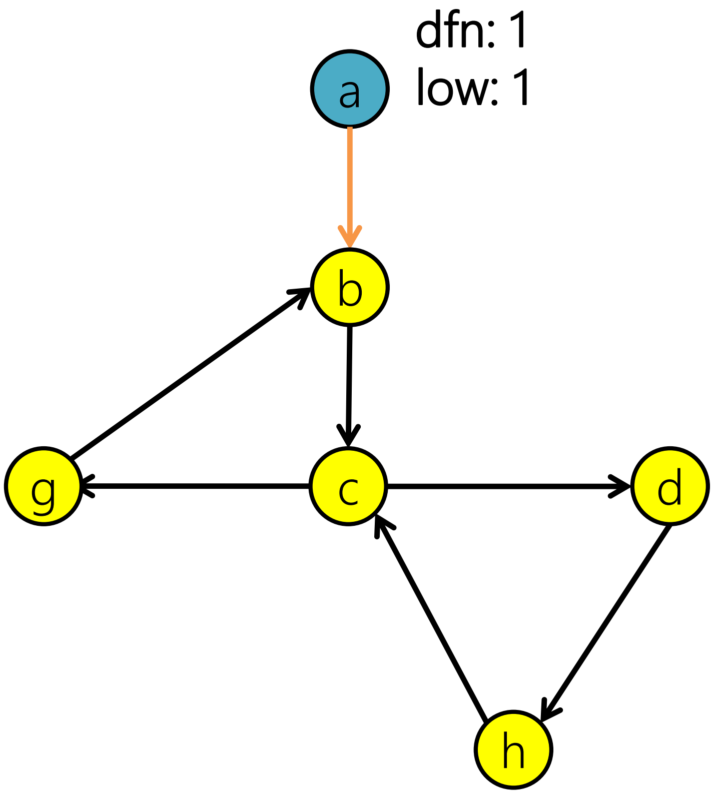


Note: This is an undirected graph

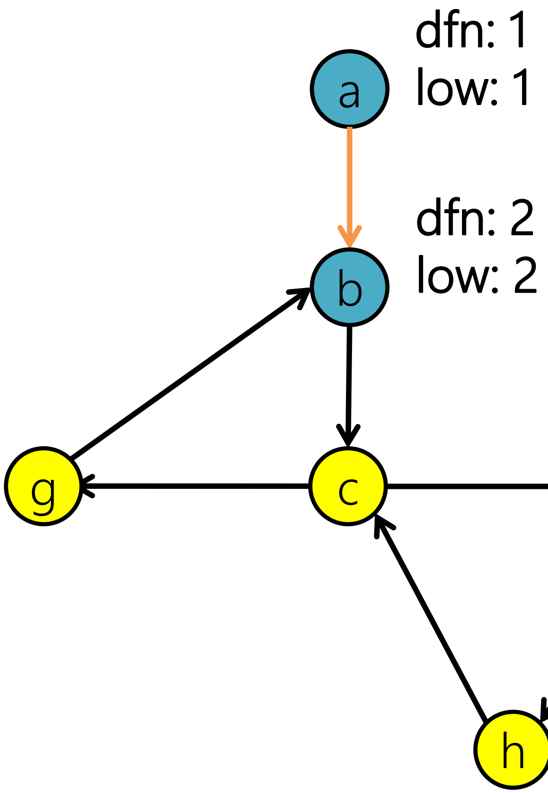
Cut vertex



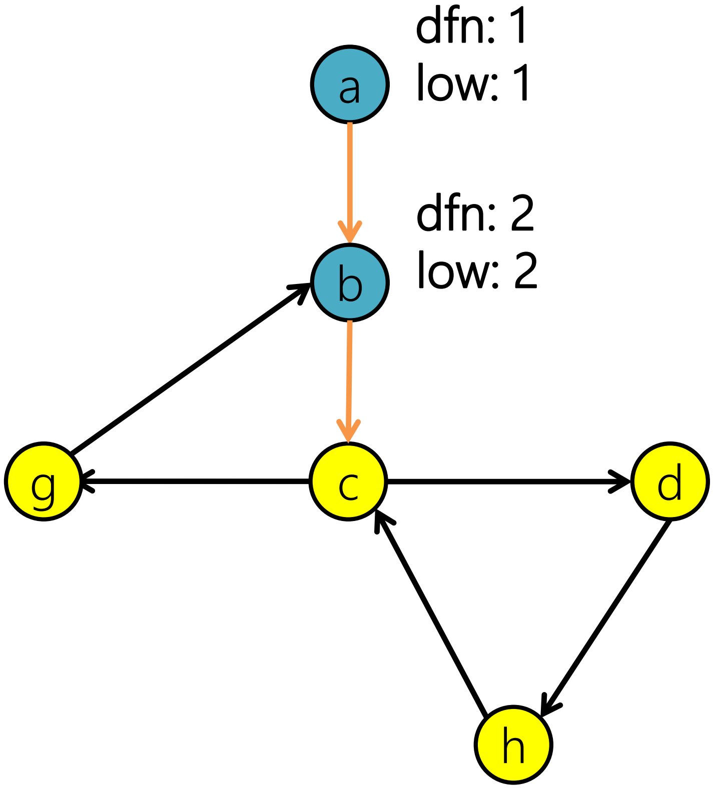
Cut vertex



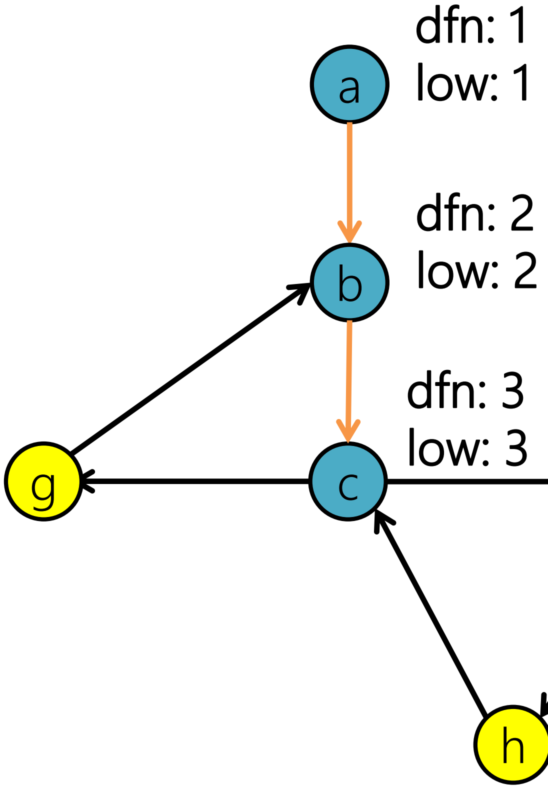
Cut vertex



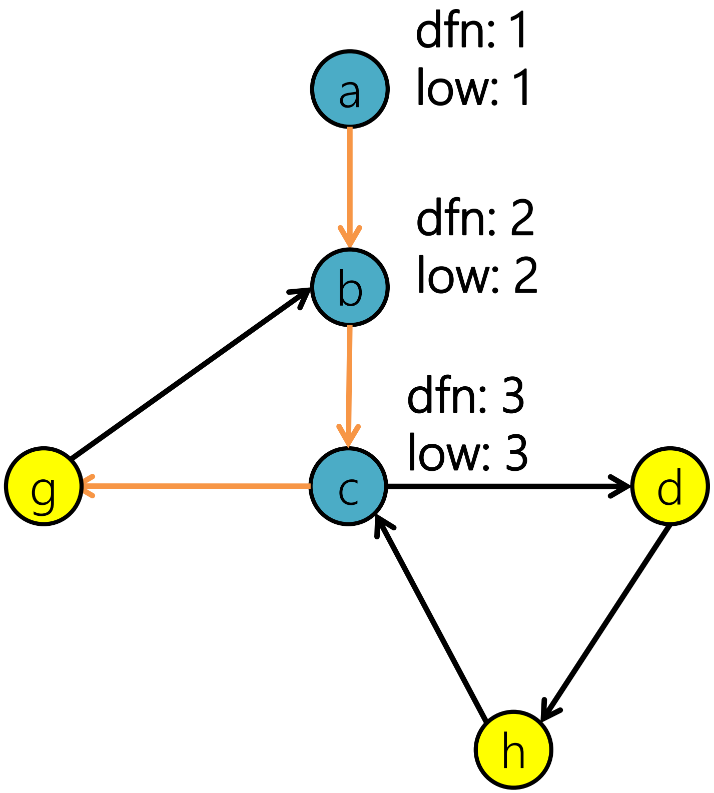
Cut vertex



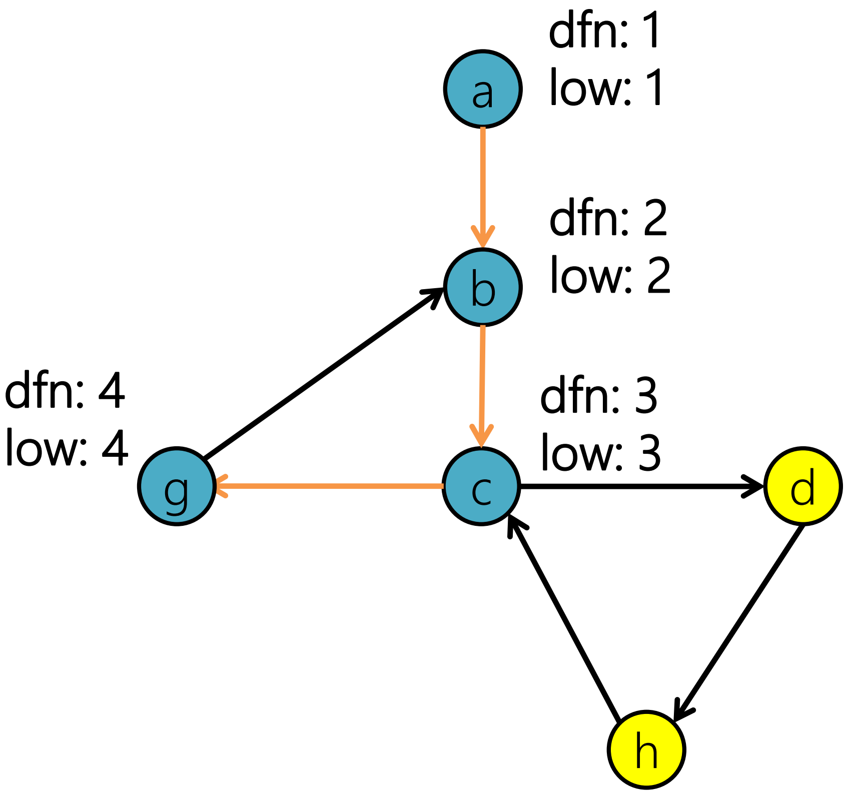
Cut vertex



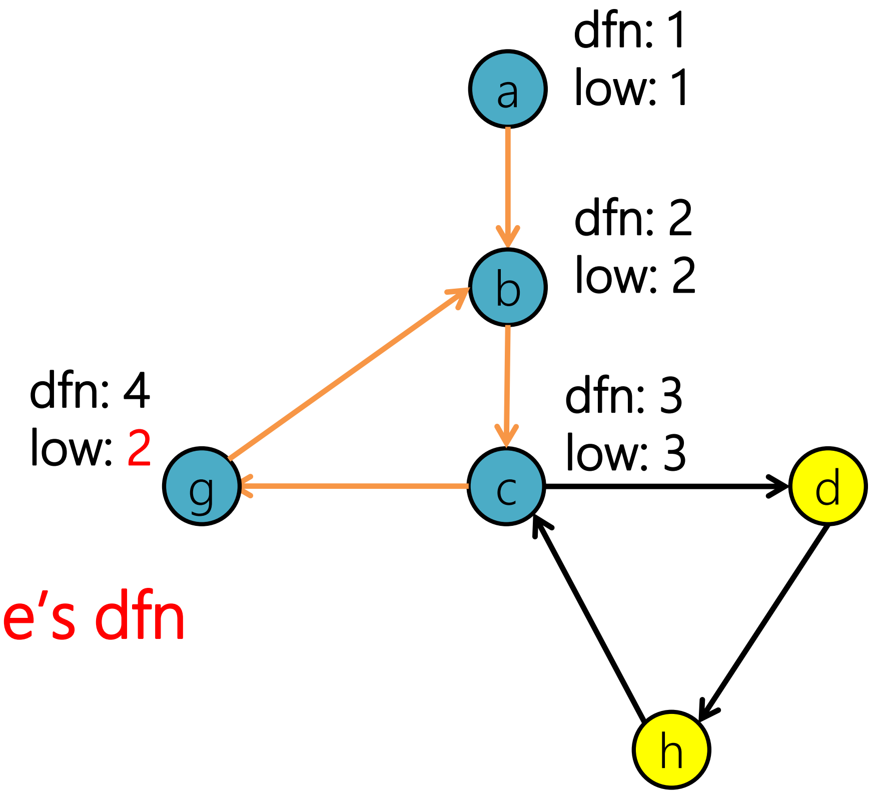
Cut vertex



Cut vertex

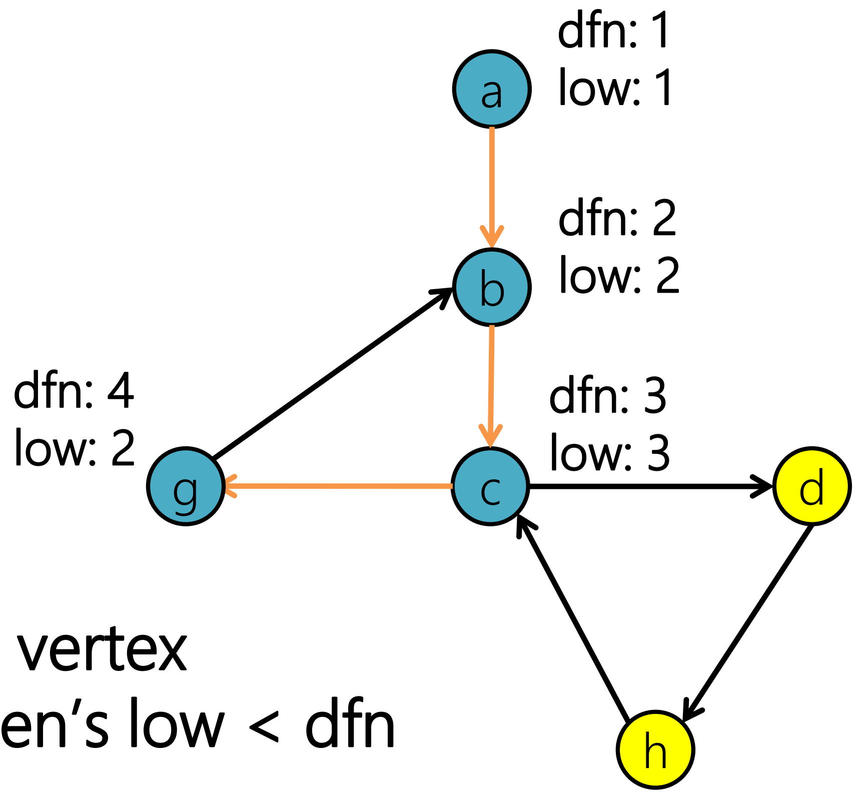


Cut vertex



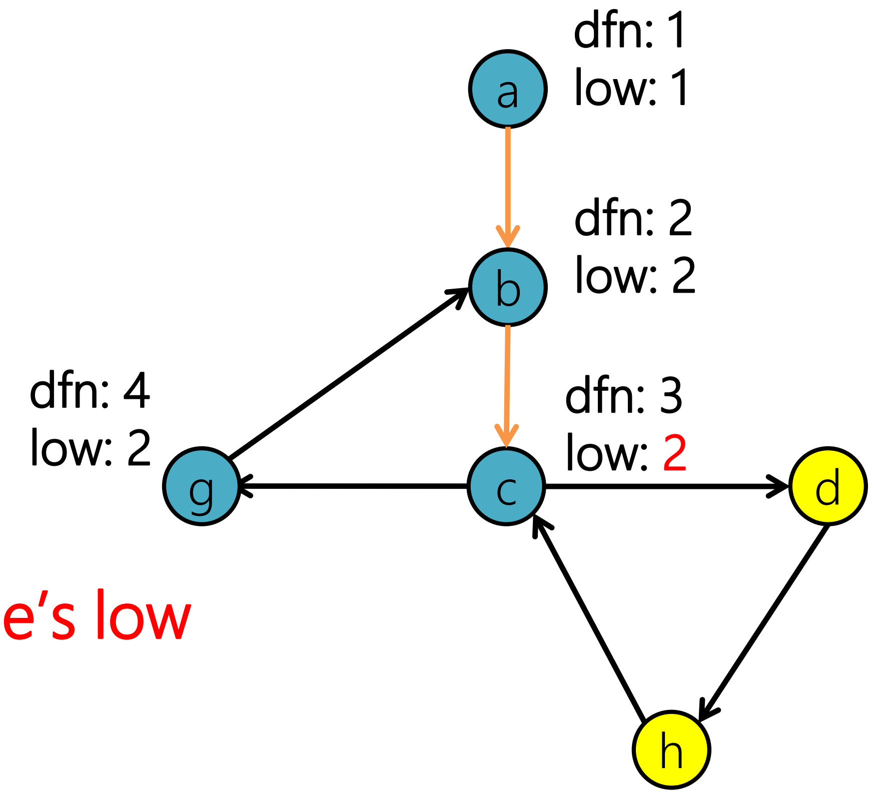
child node’s dfn

Cut vertex



g is not cut vertex
since children's low < dfn

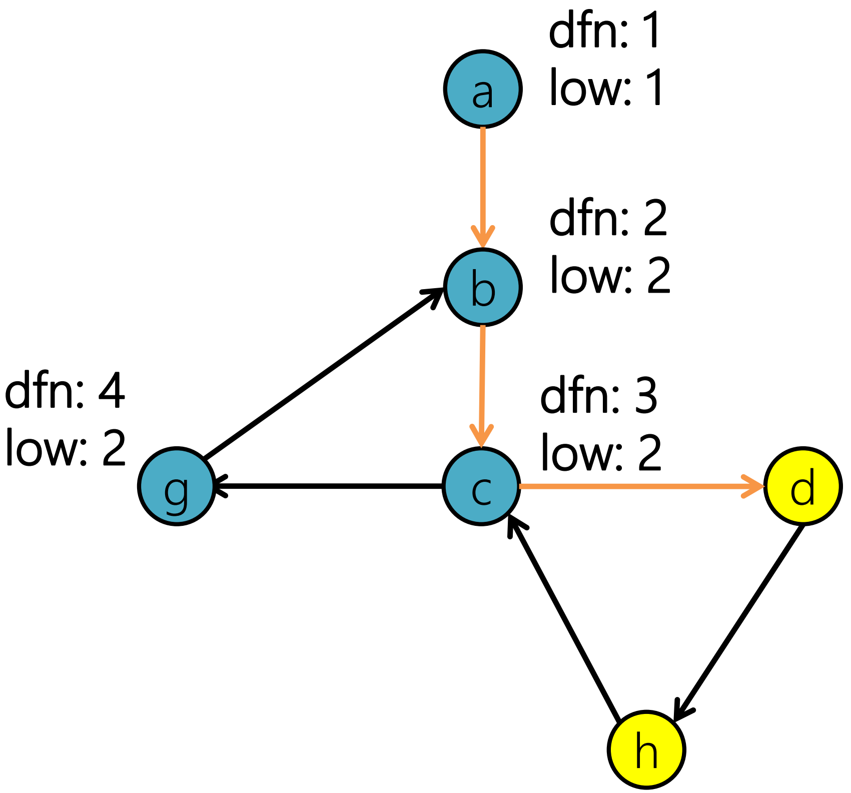
Cut vertex



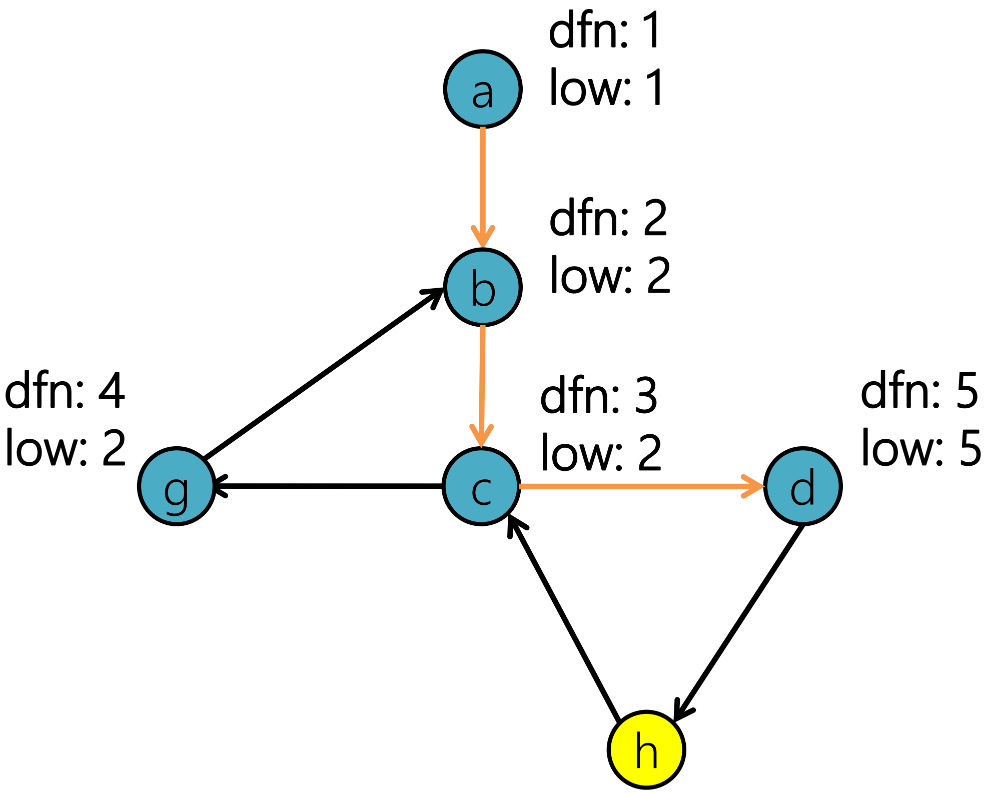
child node’s low

made by riljian

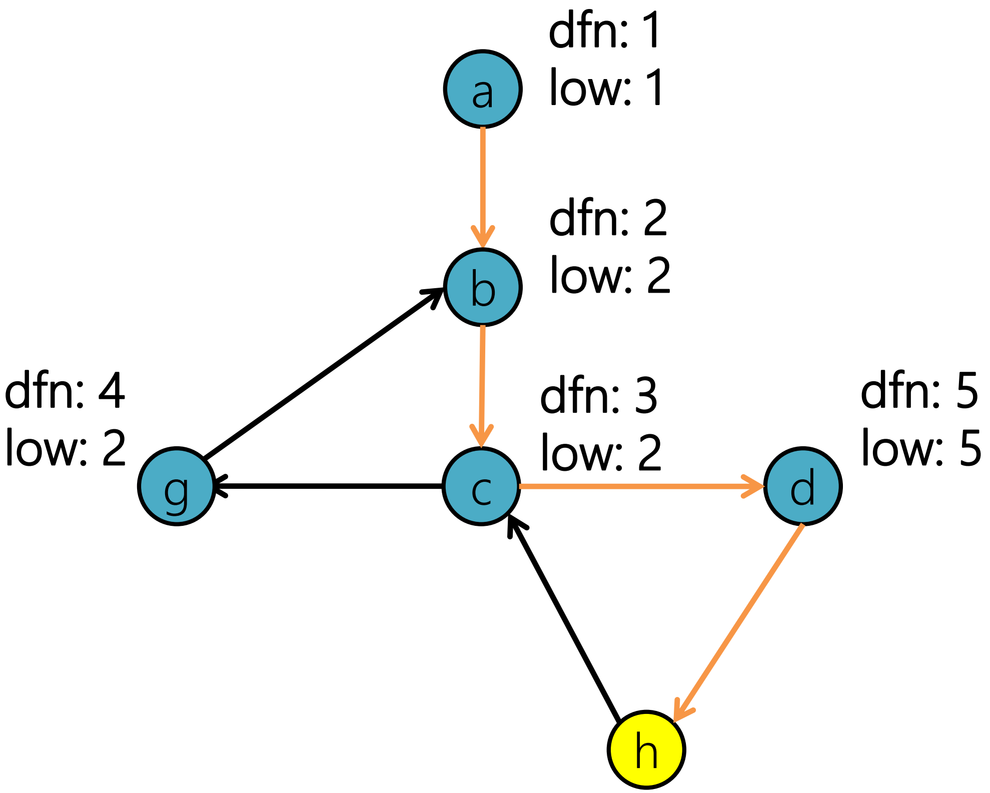
Cut vertex



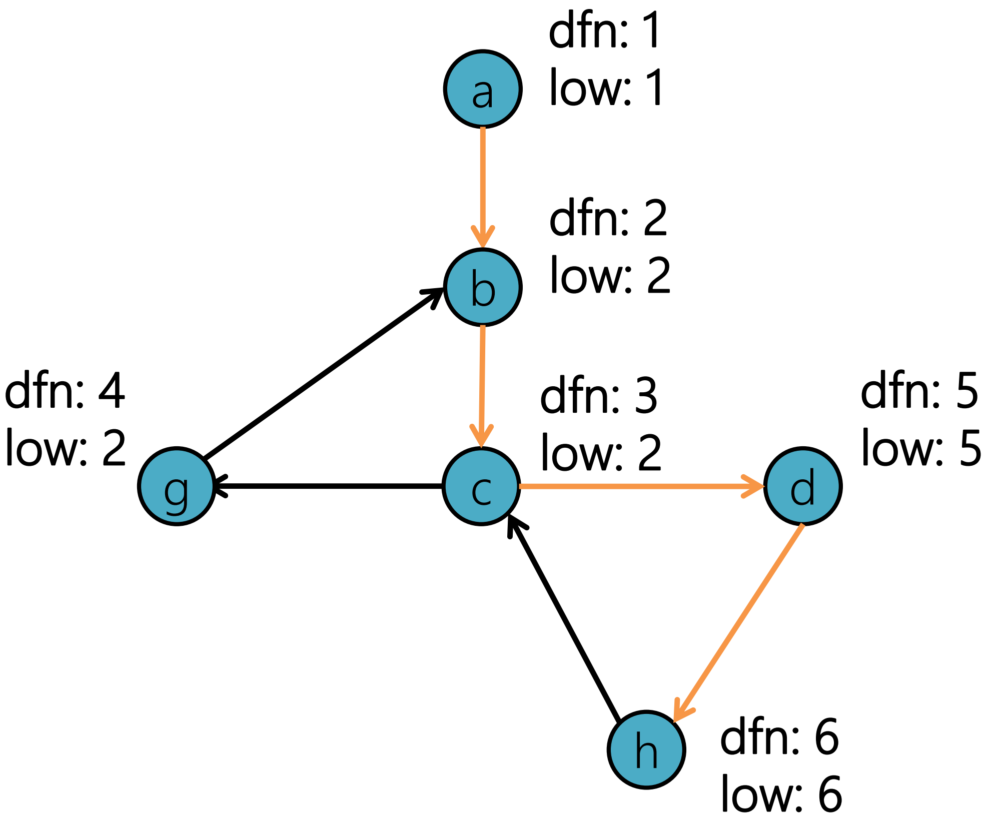
Cut vertex



Cut vertex

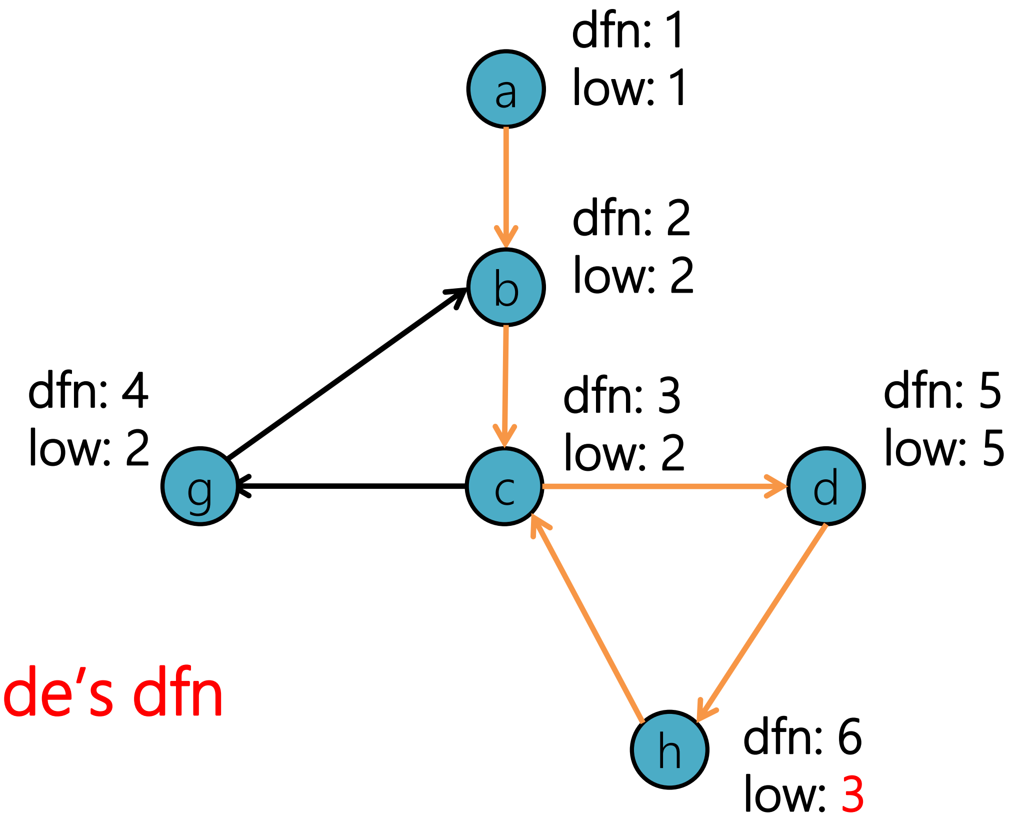


Cut vertex



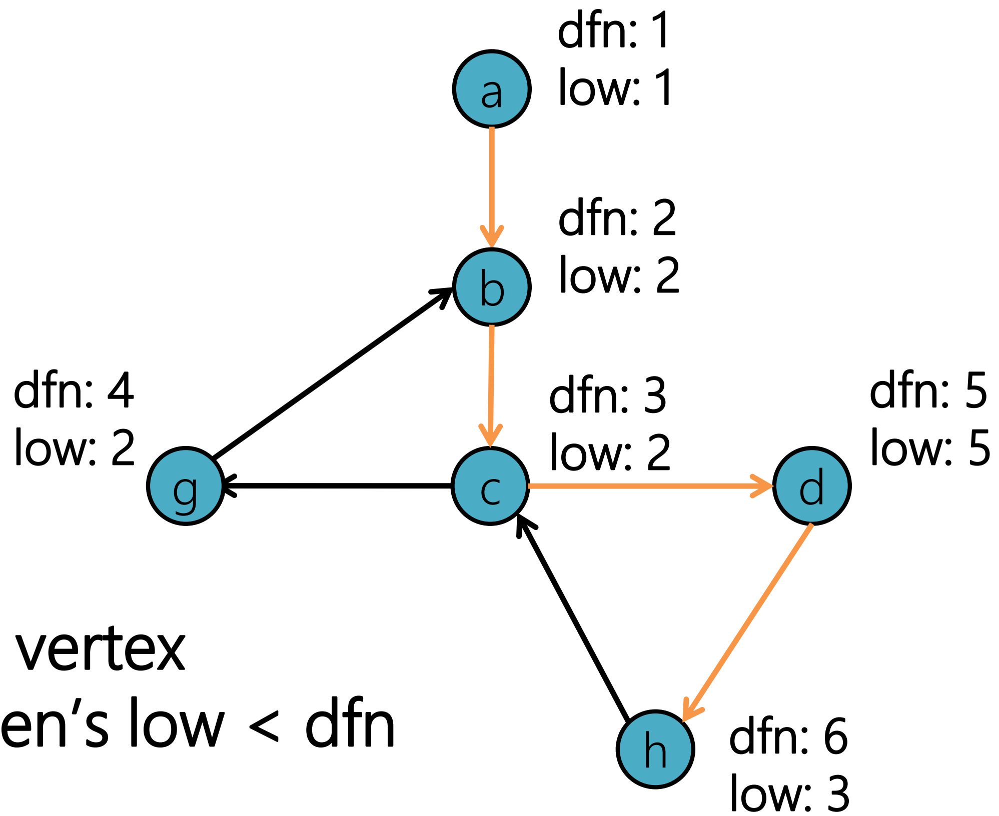
a
dfn: 1
low: 1
b
dfn: 2
low: 2
dfn: 4
low: 2
g
c
dfn: 3
low: 2
d
dfn: 5
low: 5
h
dfn: 6
low: 6

Cut vertex



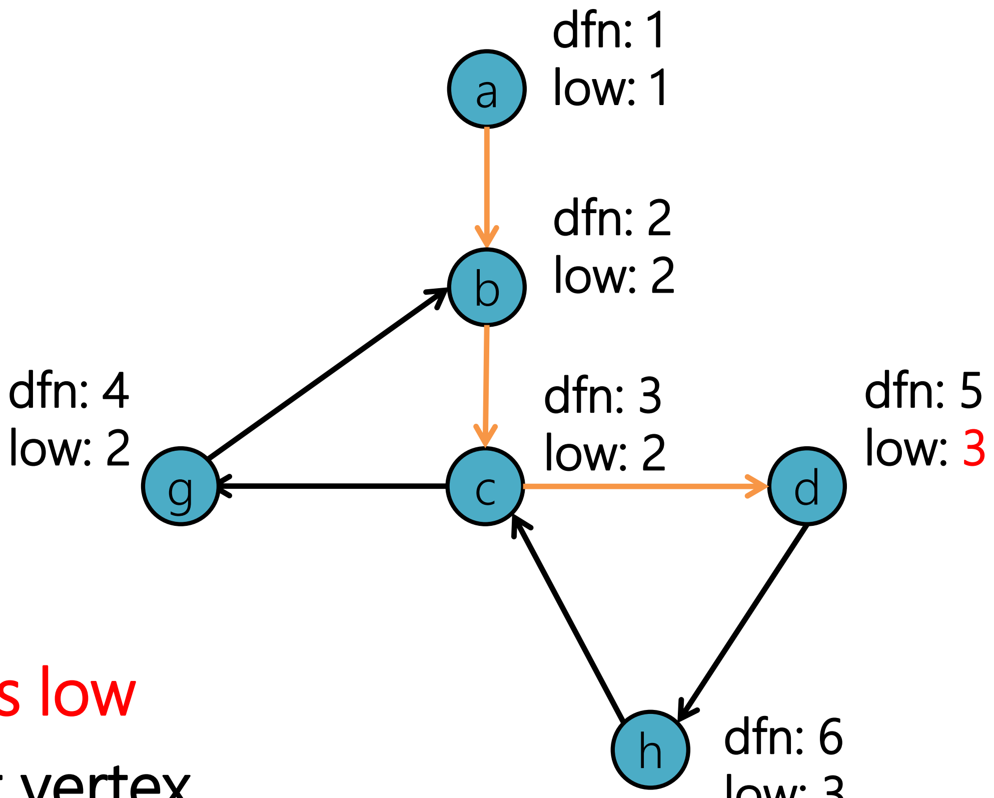
!!! child node’s dfn

Cut vertex



h is not cut vertex
since children’s low < dfn

Cut vertex

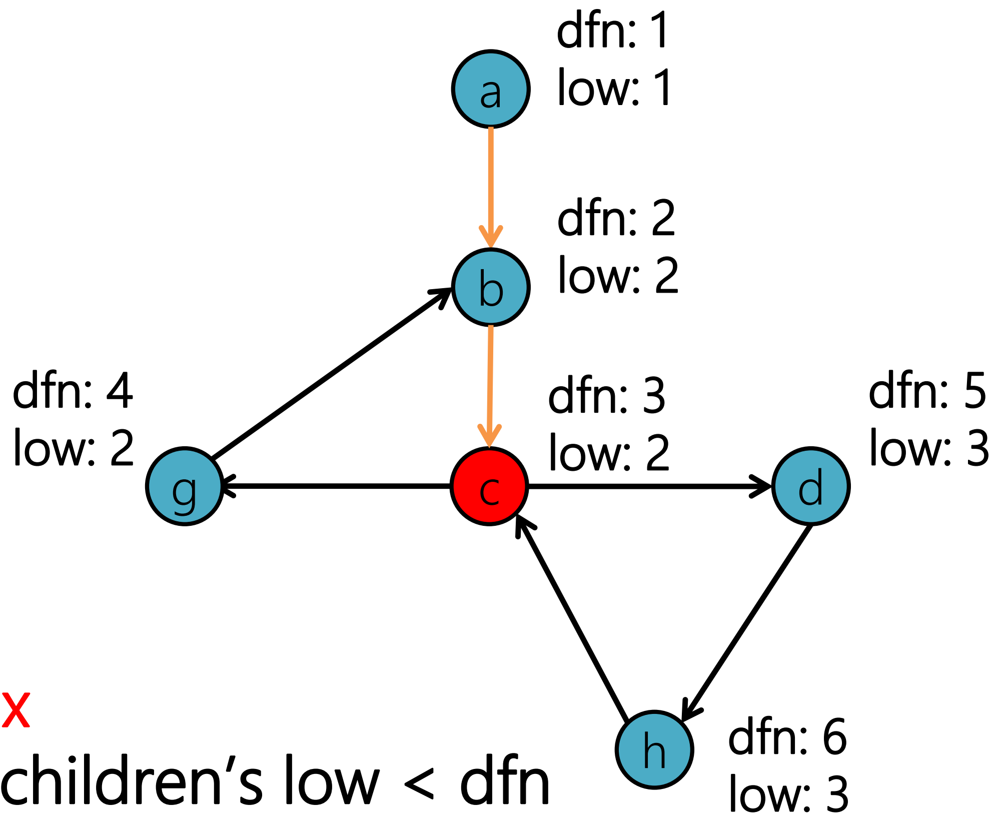


child node's low

d is not cut vertex
since children's low < dfn

made by riljian

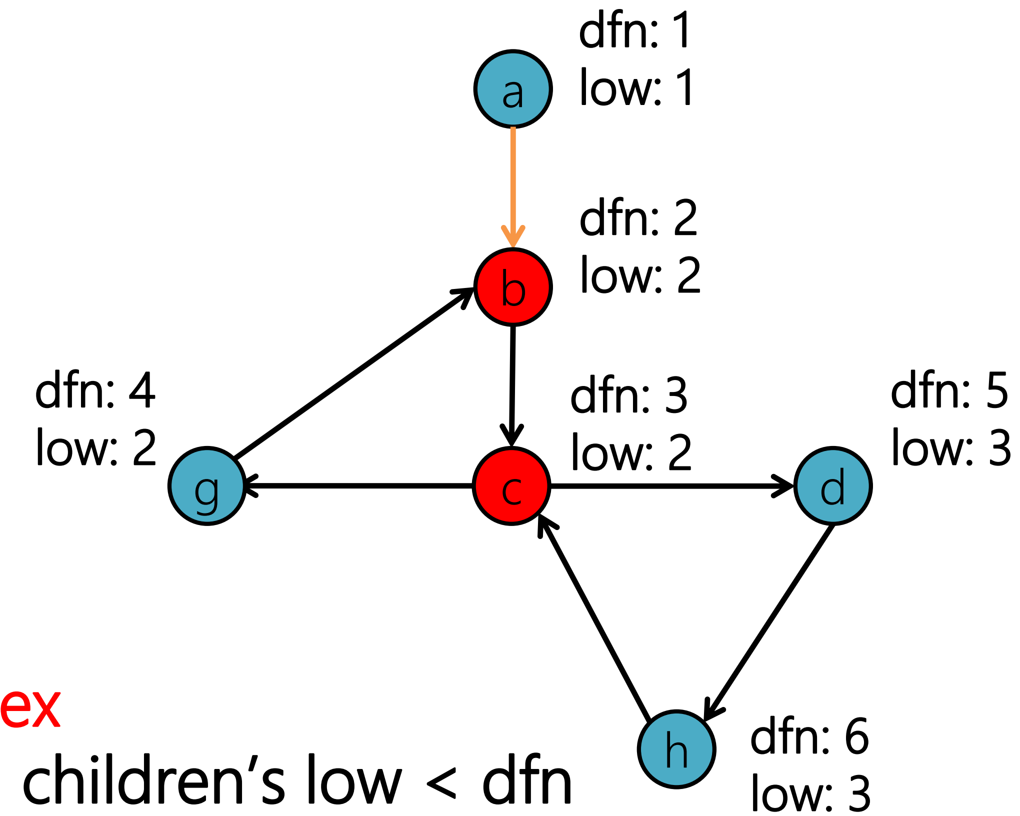
Cut vertex



c is cut vertex
since not all children’s low < dfn

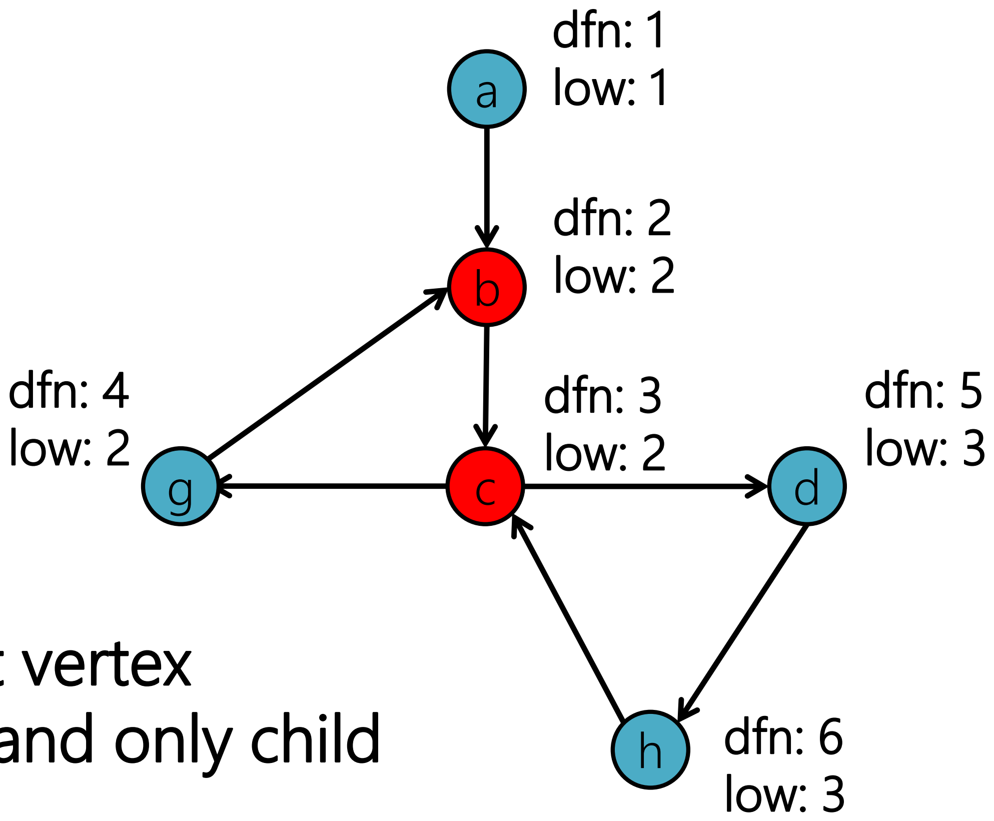
made by riljian

Cut vertex



d is cut vertex
since not all children’s low < dfn

Cut vertex



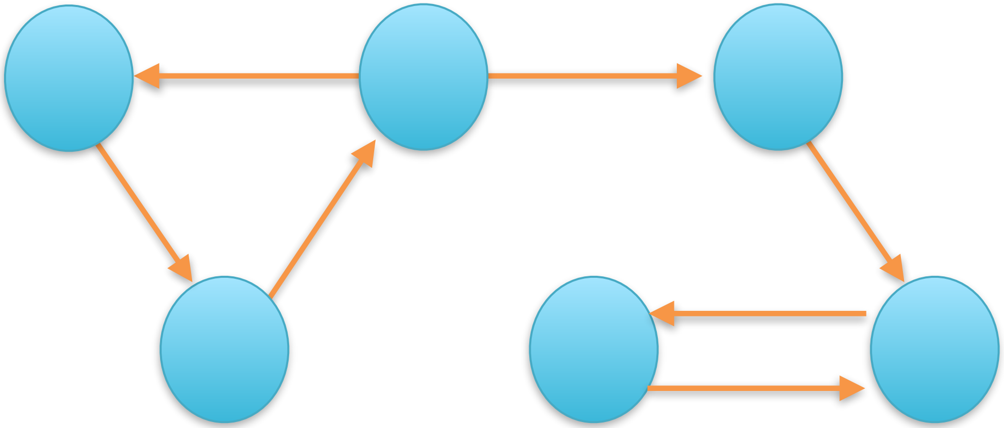
a is not cut vertex
since root and only child

Practice

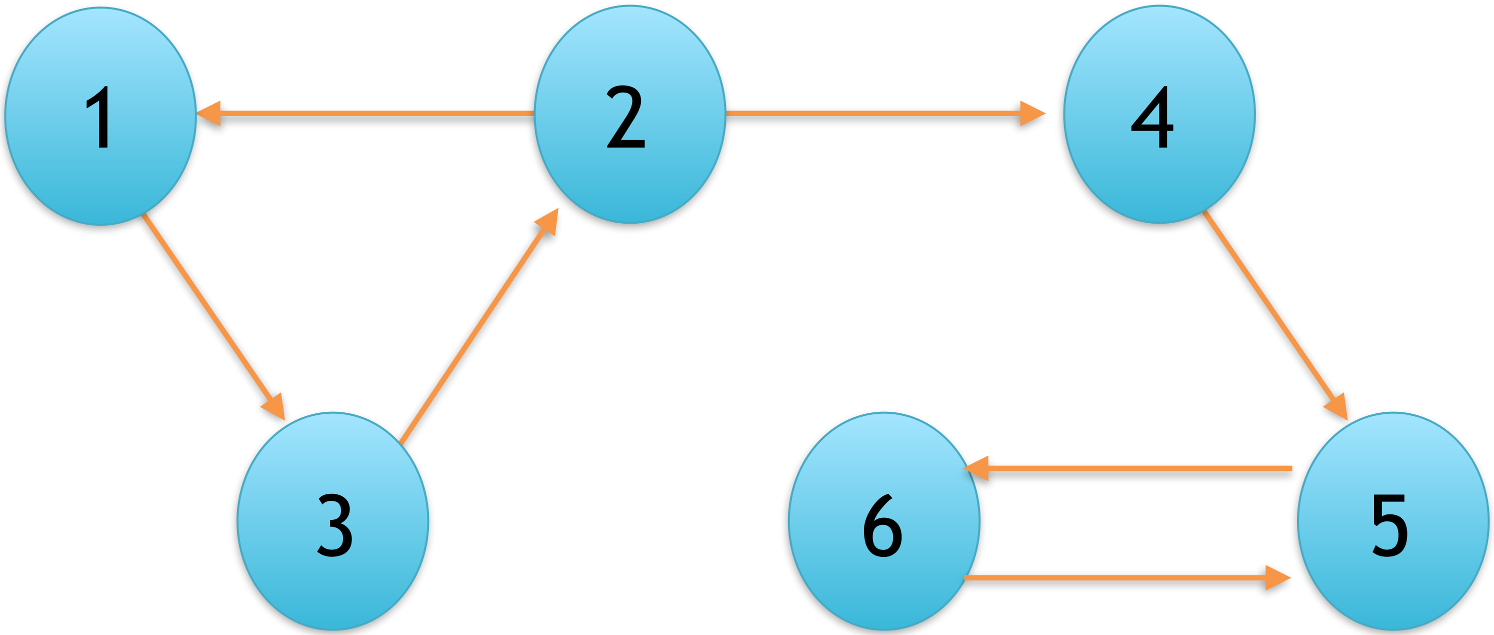
UVA - 315

SCC

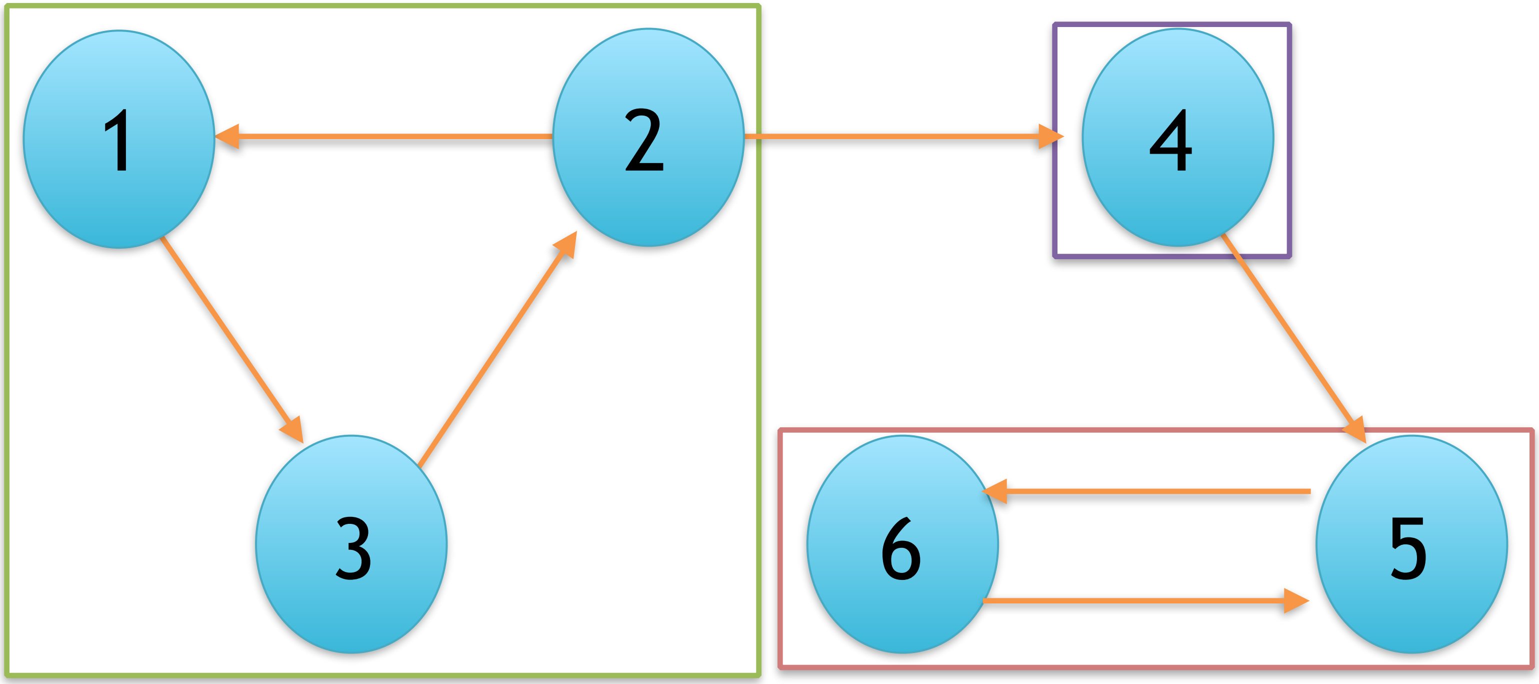
- connected component in directed graph
  - same definition in undirected graph



SCC



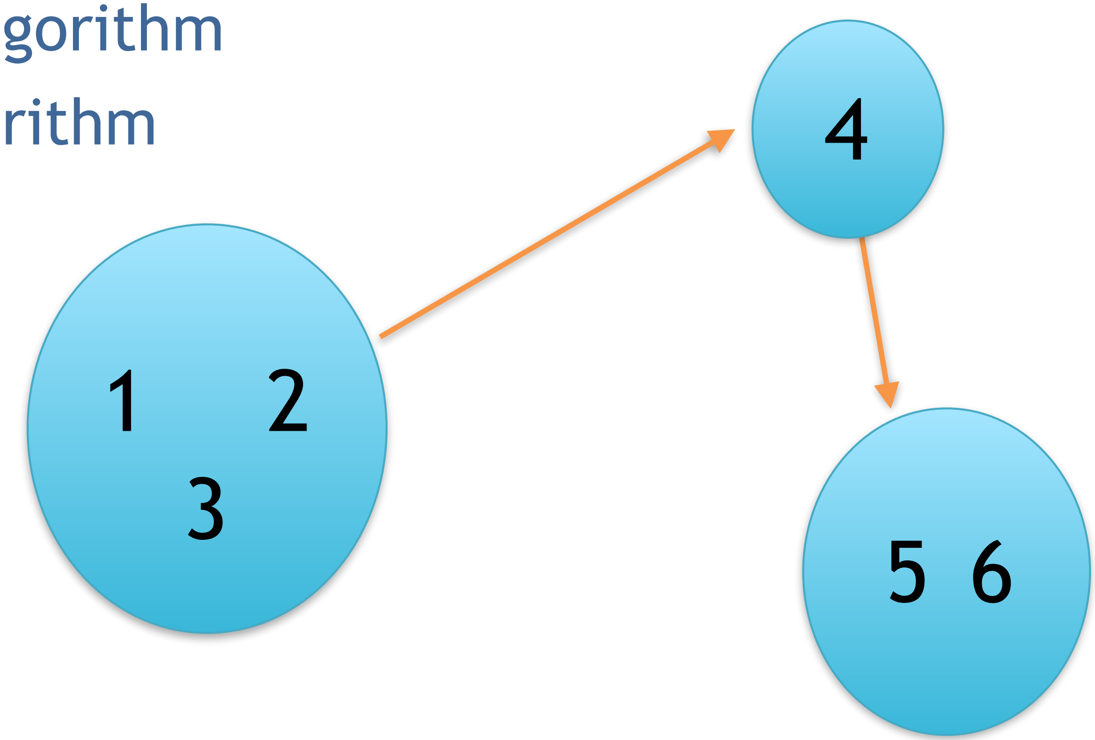
SCC



SCC

find all SCCs, contract all cycles —> DAG (directed acyclic graph)

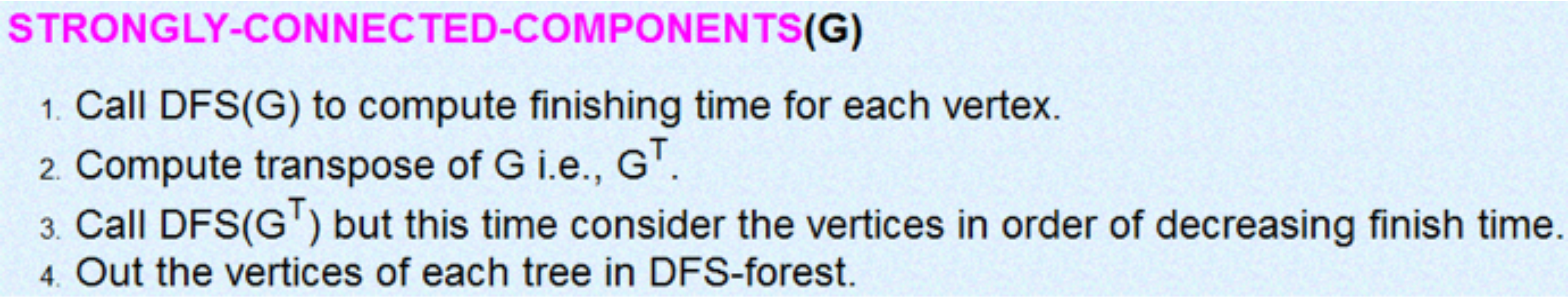
- Kosaraju's Algorithm
- Tarjan's Algorithm



made by tommy5198

SCC

- Kosaraju's algorithm

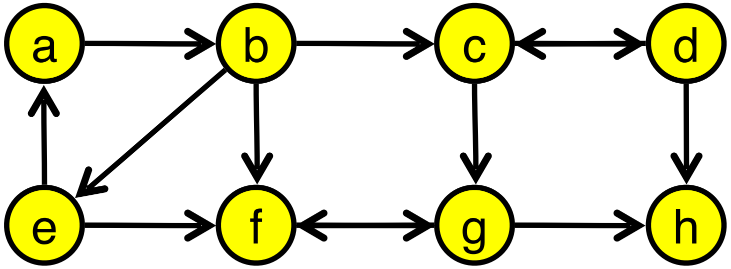


```
STRONGLY-CONNECTED-COMPONENTS(G)
1. Call DFS(G) to compute finishing time for each vertex.
2. Compute transpose of G i.e., $G^T$.
3. Call DFS($G^T$) but this time consider the vertices in order of decreasing finish time.
4. Out the vertices of each tree in DFS-forest.
```

twice DFS —> total complexity: O(V+E)

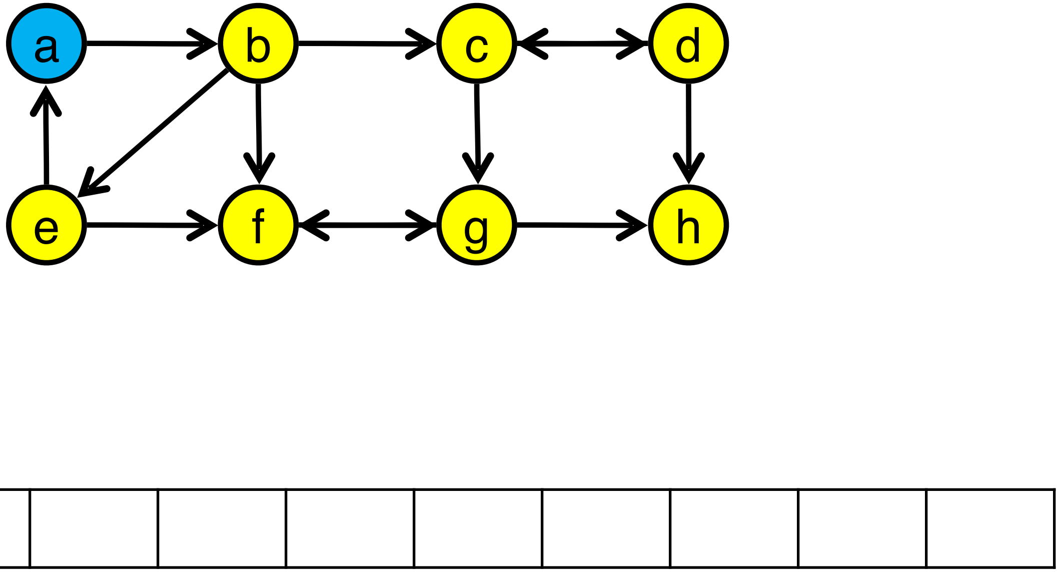
SCC

- Algorithm



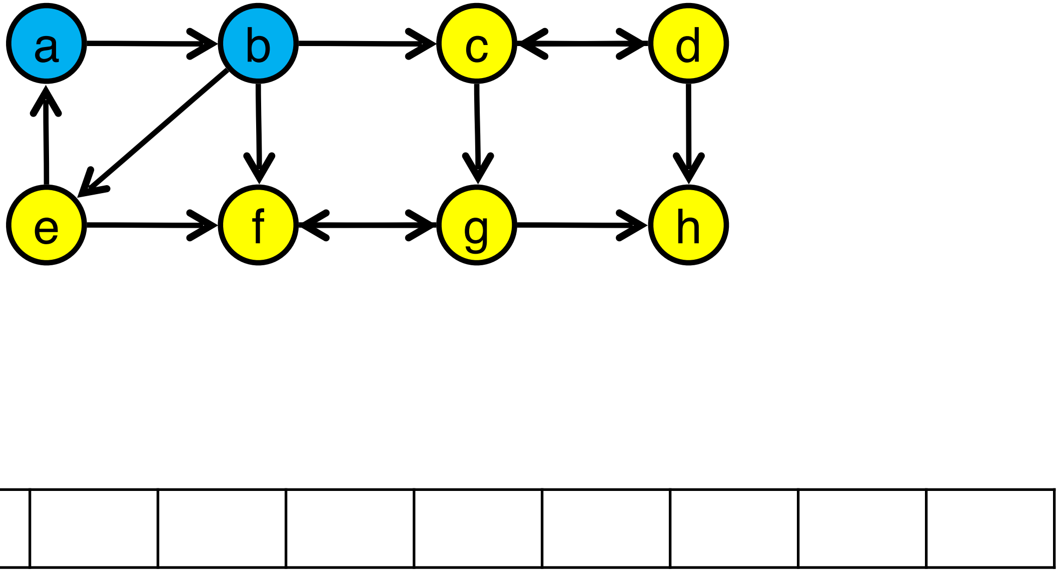
SCC

- Algorithm



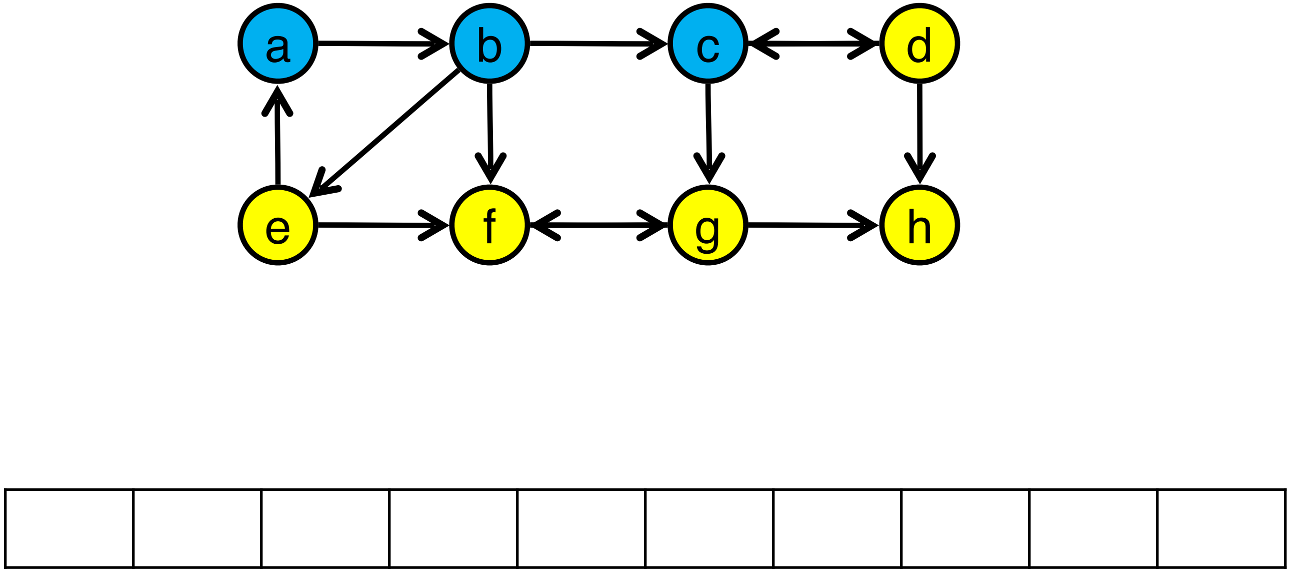
SCC

- Algorithm



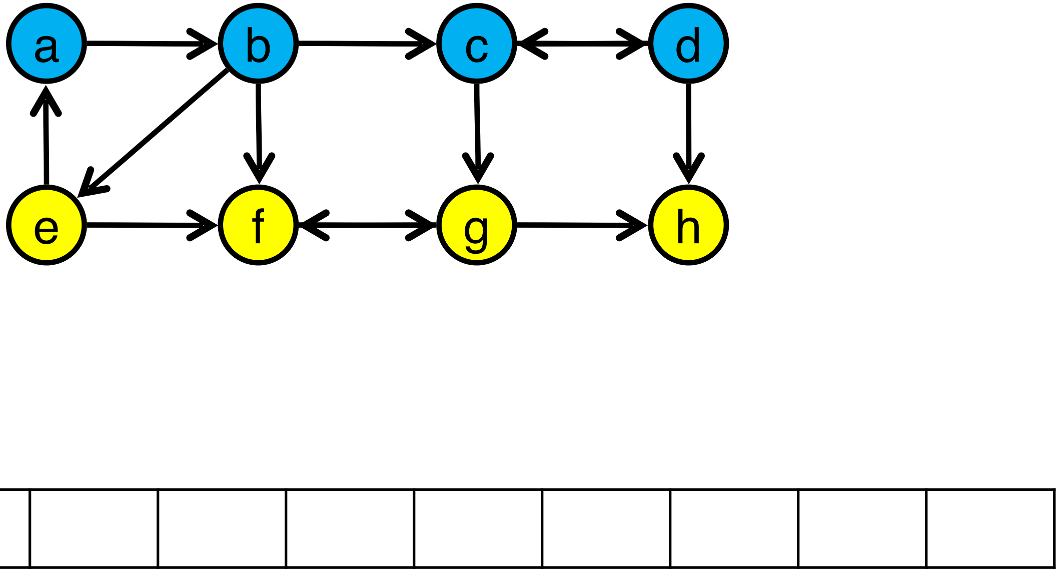
SCC

- Algorithm



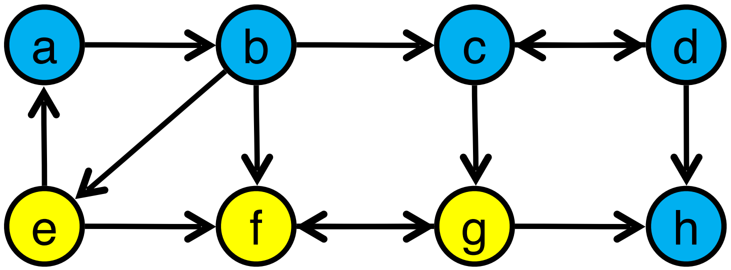
SCC

- Algorithm



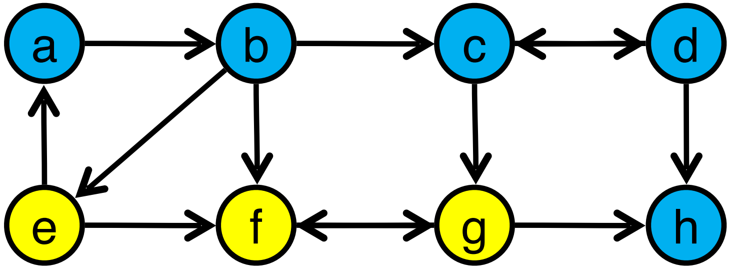
SCC

- Algorithm



SCC

- Algorithm

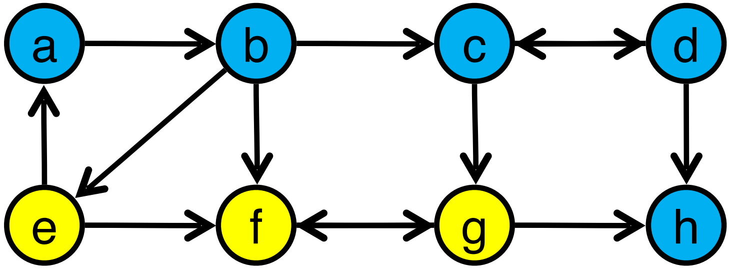


<— h finish!

h									

SCC

- Algorithm

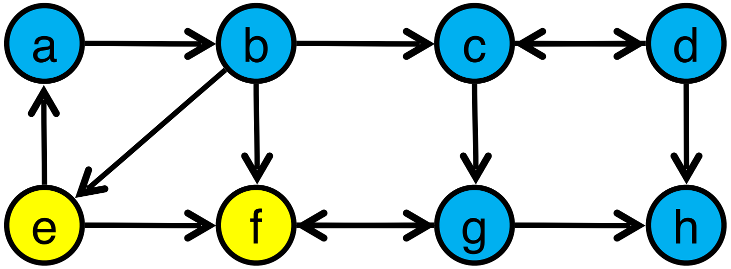


<— d finish!

h	d								

SCC

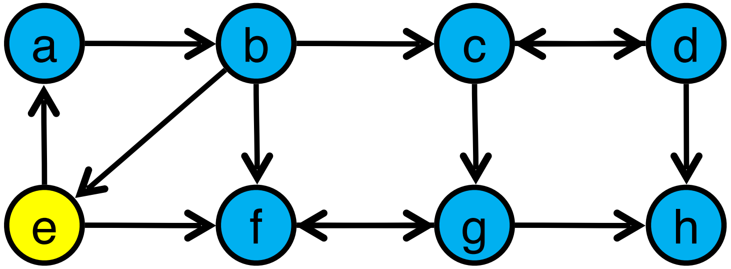
- Algorithm



h	d								

SCC

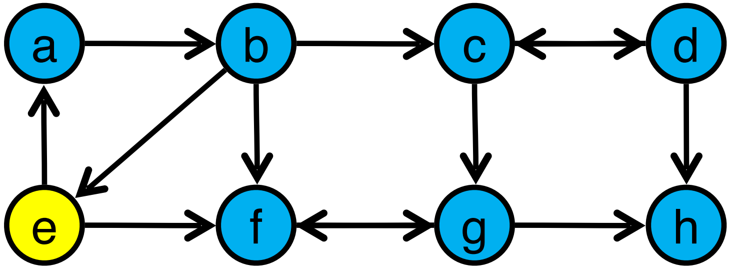
- Algorithm



h	d								

SCC

- Algorithm

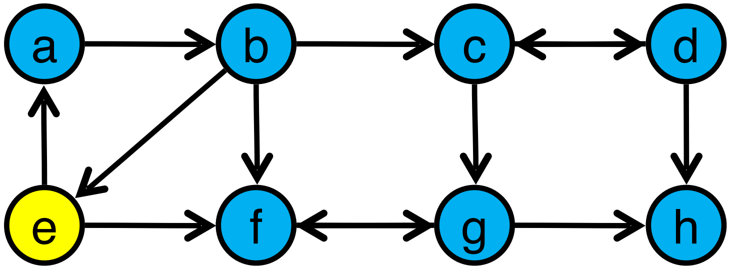


^ f finish!

h	d	f							

SCC

- Algorithm



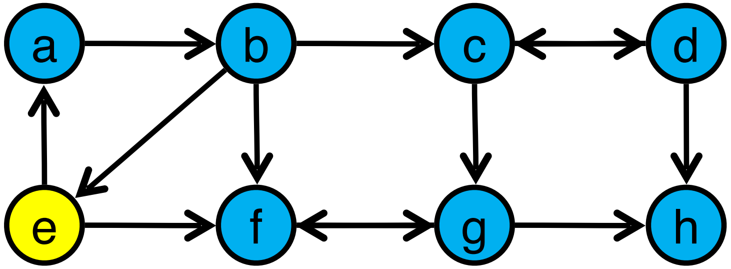
^ g finish!

h	d	f	g						

SCC

- Algorithm

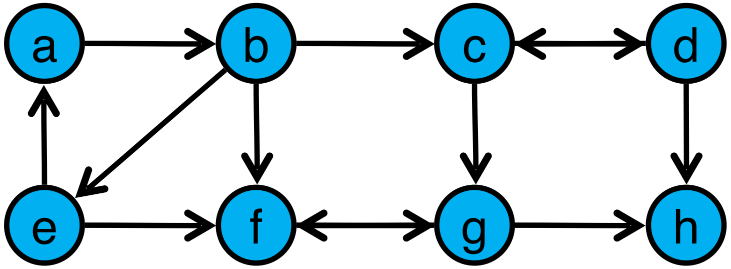
c finish!



h	d	f	g	c					

SCC

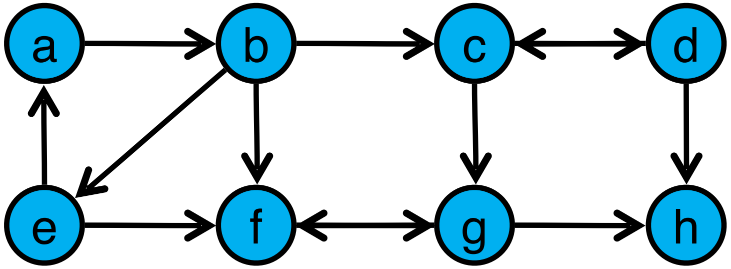
- Algorithm



h	d	f	g	c					

SCC

- Algorithm

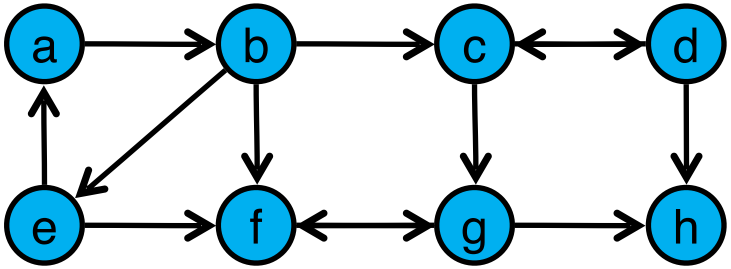


^ e finish!

h	d	f	g	c	e				

SCC

- Algorithm



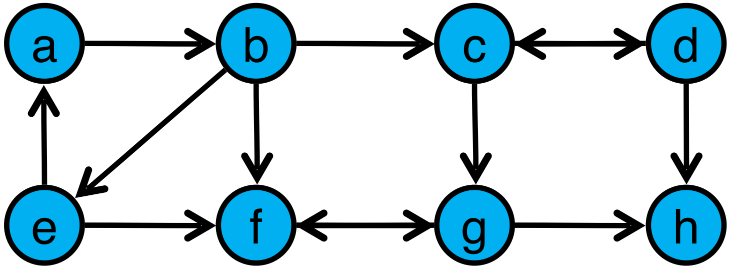
b finish!

h	d	f	g	c	e	b			

SCC

- Algorithm

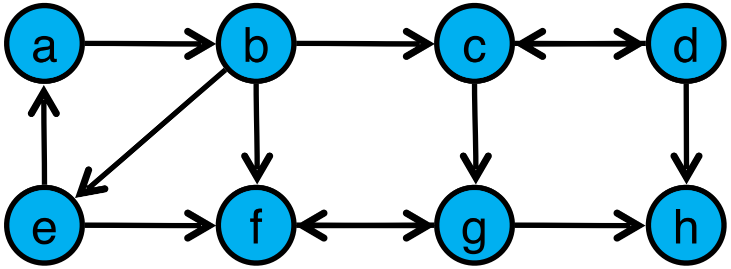
a finish!



h	d	f	g	c	e	b	a		

SCC

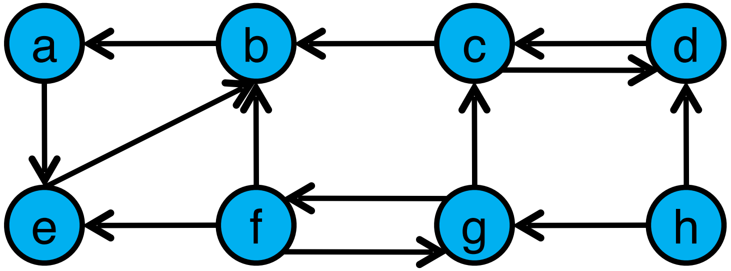
- Algorithm
  - Reverse the graph



h	d	f	g	c	e	b	a		

SCC

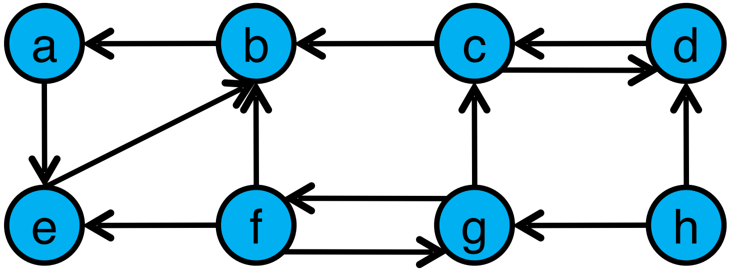
- Algorithm
  - Reverse the graph



h	d	f	g	c	e	b	a		

SCC

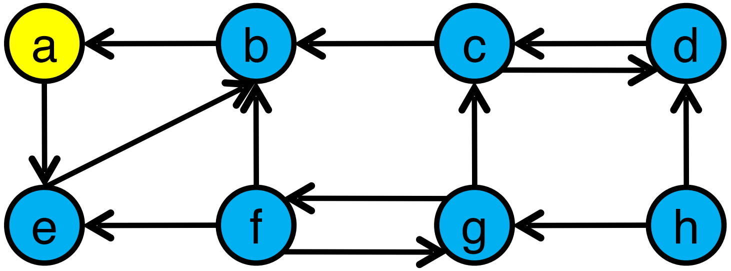
- Algorithm
  - Reverse the graph
  - Re-search by the ending time



h	d	f	g	c	e	b	a		

SCC

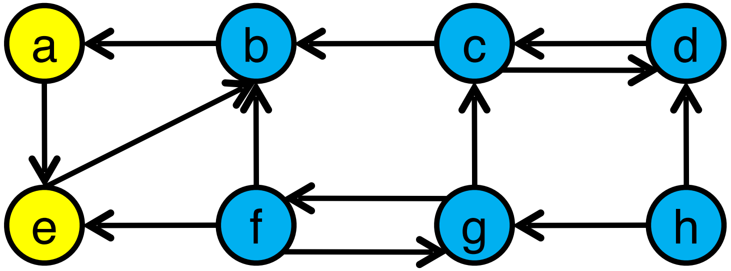
- Algorithm
  - Reverse the graph
  - Re-search by the ending time



h	d	f	g	c	e	b	a		

SCC

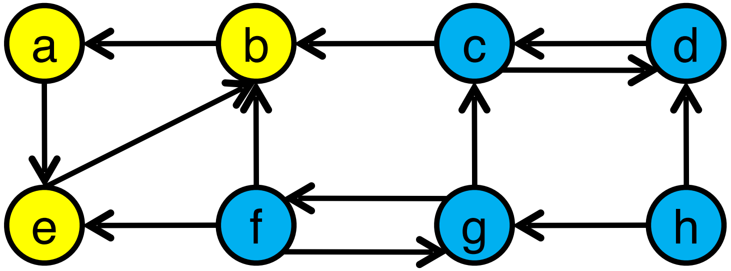
- Algorithm
  - Reverse the graph
  - Re-search by the ending time



h	d	f	g	c	e	b	a		

SCC

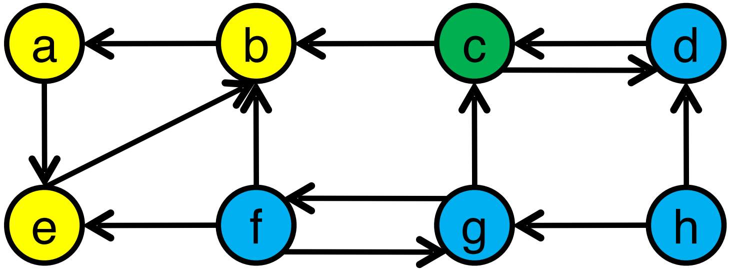
- Algorithm
  - Reverse the graph
  - Re-search by the ending time



h	d	f	g	c	e	b	a		

SCC

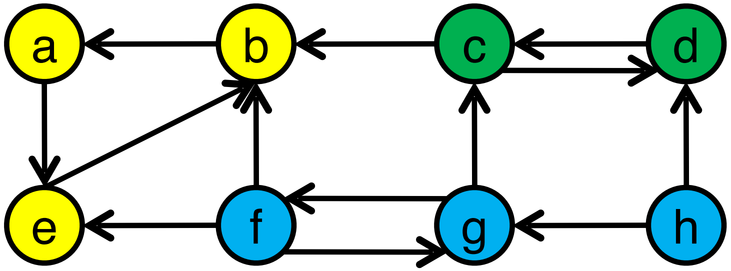
- Algorithm
  - Reverse the graph
  - Re-search by the ending time



h	d	f	g	c	e	b	a		

SCC

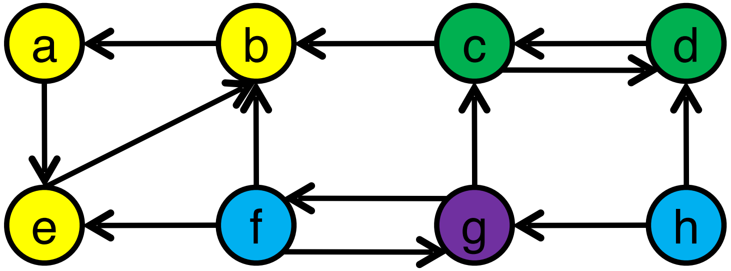
- Algorithm
  - Reverse the graph
  - Re-search by the ending time



h	d	f	g	c	e	b	a		

SCC

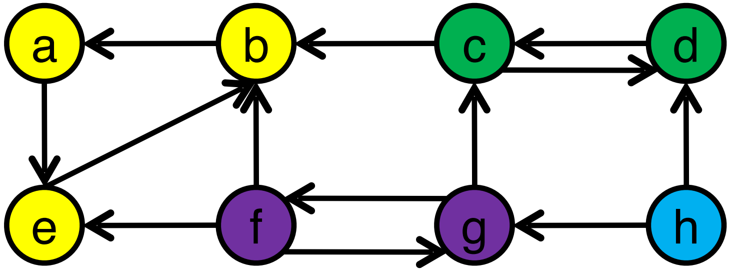
- Algorithm
  - Reverse the graph
  - Re-search by the ending time



h	d	f	g	c	e	b	a		

SCC

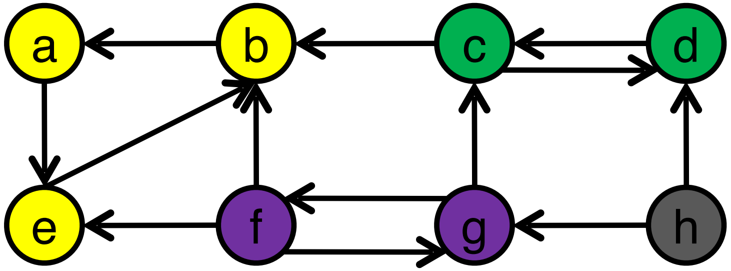
- Algorithm
  - Reverse the graph
  - Re-search by the ending time



h	d	f	g	c	e	b	a		

SCC

- Algorithm
  - Reverse the graph
  - Re-search by the ending time

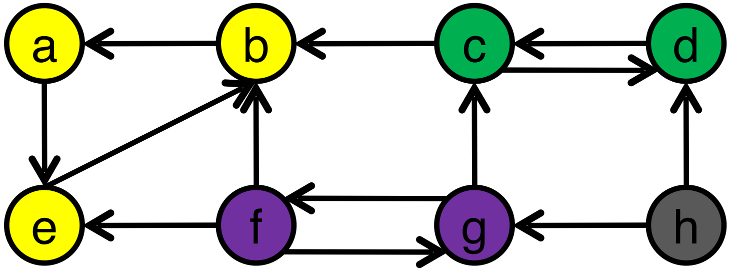


h	d	f	g	c	e	b	a		

SCC

- Algorithm
  - Reverse the graph
  - Re-search by the ending time

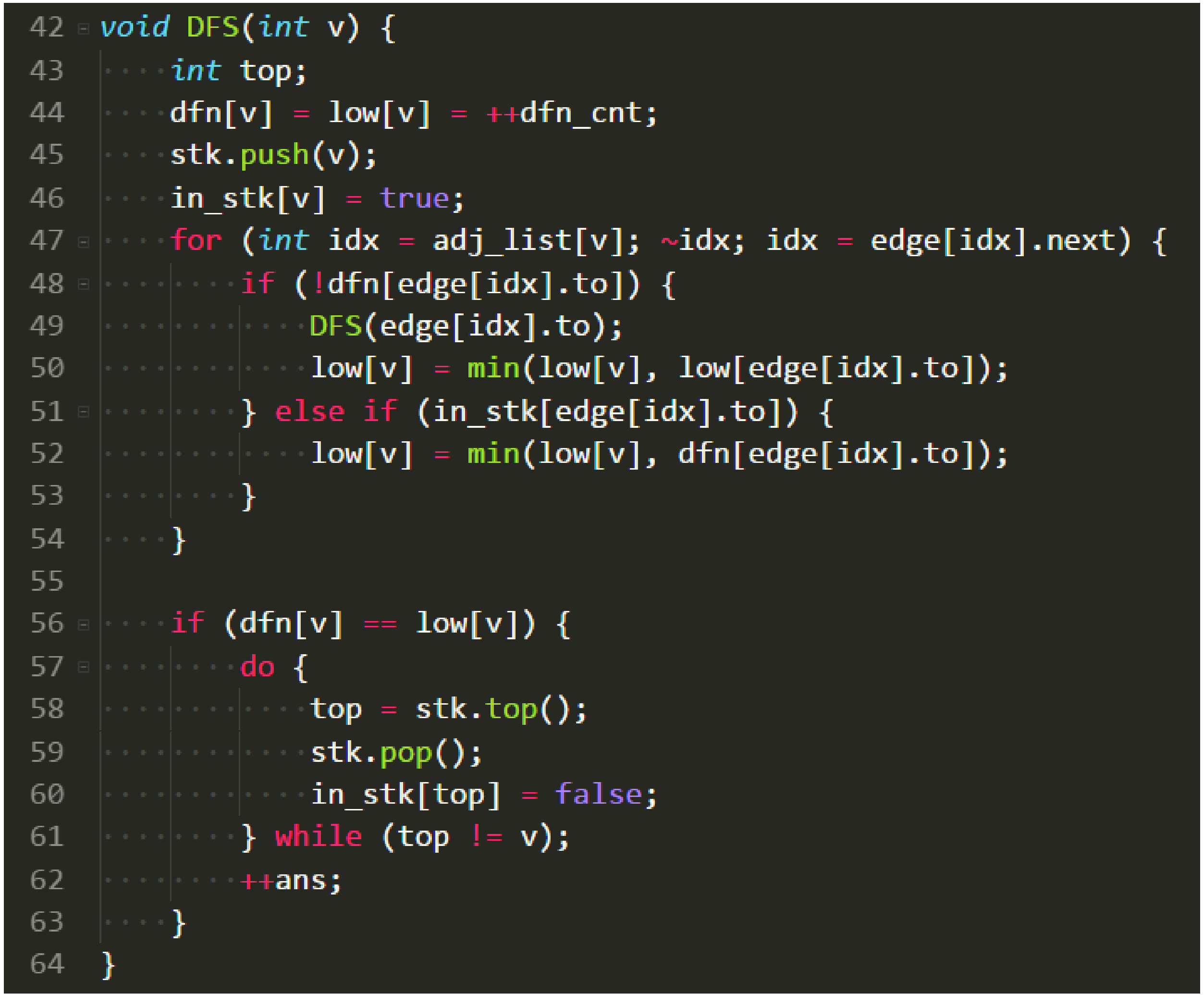
4 components



h	d	f	g	c	e	b	a		

SCC

- Tarjan

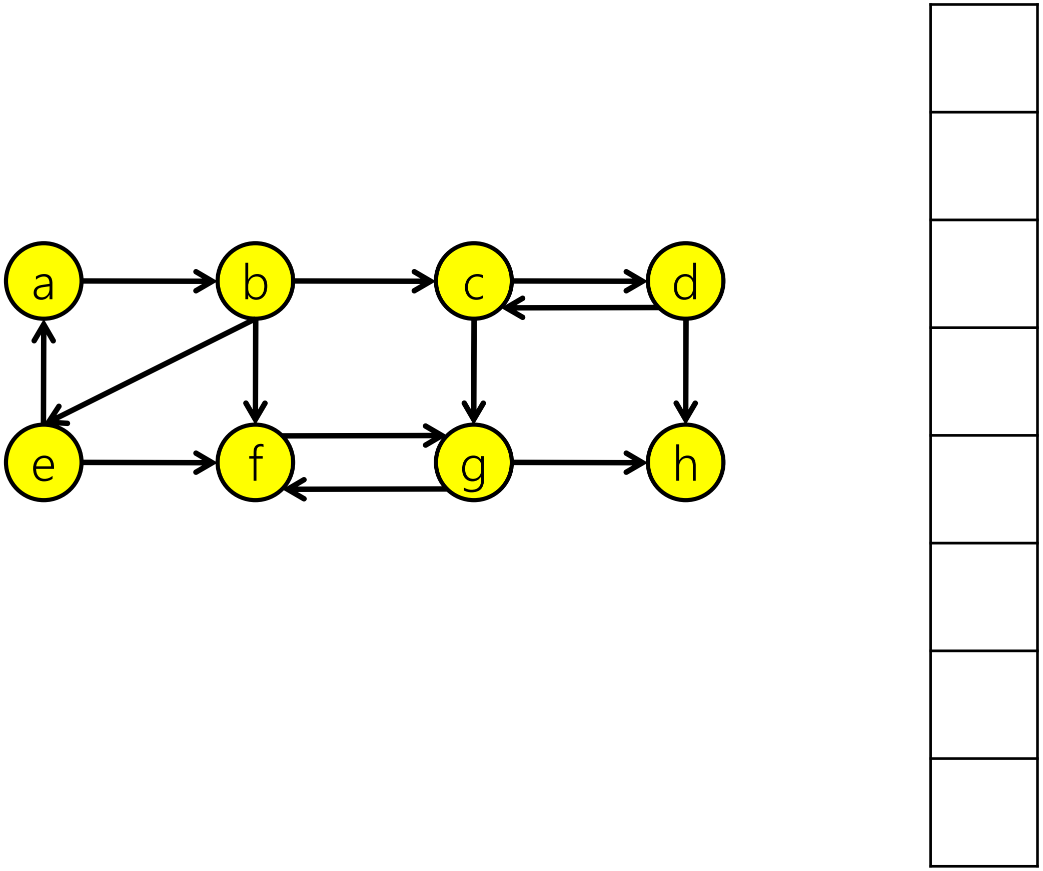


```
void DFS(int v) {
    int top;
    dfn[v] = low[v] = ++dfn_cnt;
    stk.push(v);
    in_stk[v] = true;
    for (int idx = adj_list[v]; ~idx; idx = edge[idx].next) {
        if (!dfn[edge[idx].to]) {
            DFS(edge[idx].to);
            low[v] = min(low[v], low[edge[idx].to]);
        } else if (in_stk[edge[idx].to]) {
            low[v] = min(low[v], dfn[edge[idx].to]);
        }
    }

    if (dfn[v] == low[v]) {
        do {
            top = stk.top();
            stk.pop();
            in_stk[top] = false;
        } while (top != v);
        ++ans;
    }
}
```

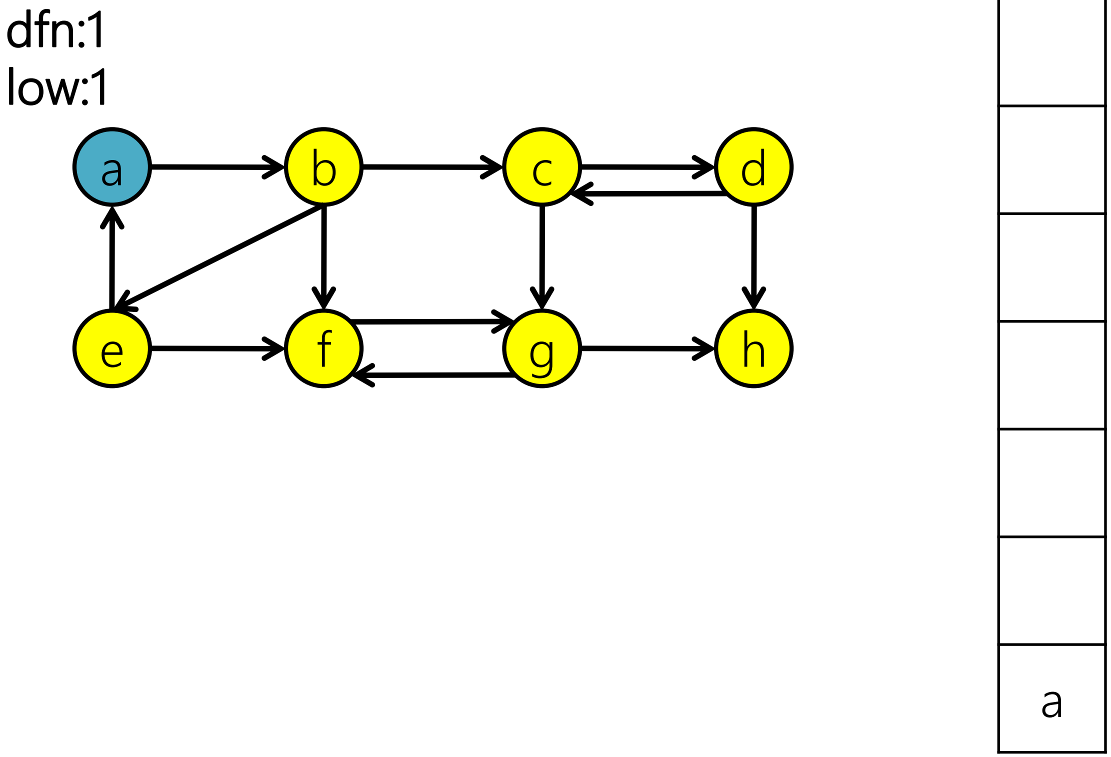
SCC

- Tarjan



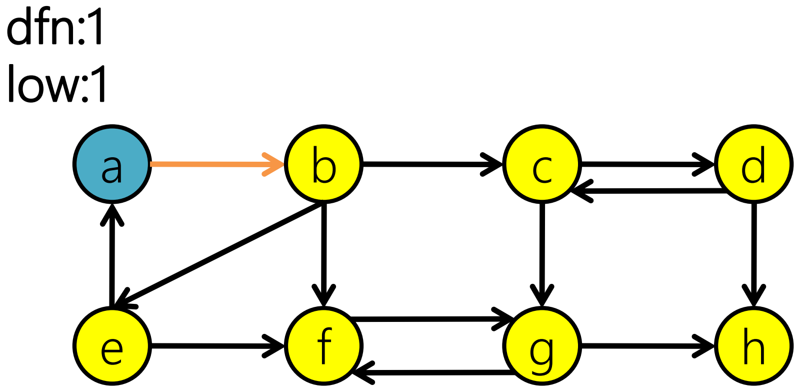
SCC

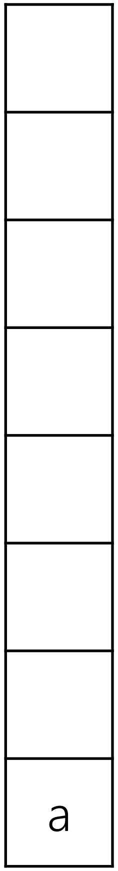
- Tarjan



SCC

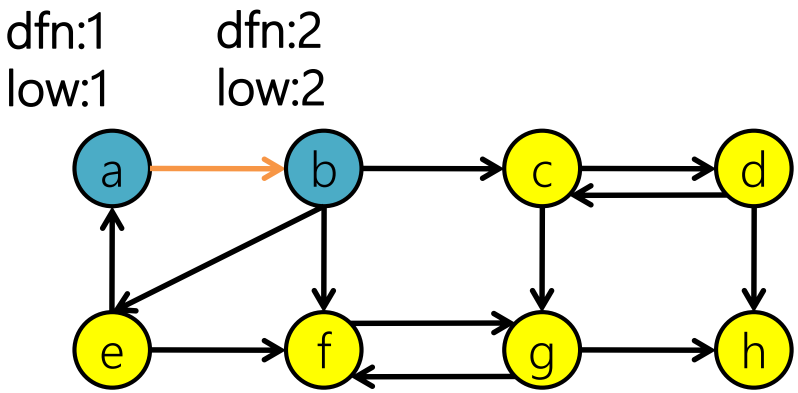
- Tarjan



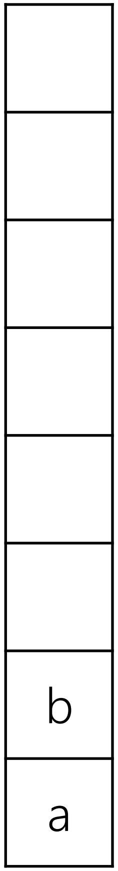


SCC

- Tarjan


dfn:1
low:1

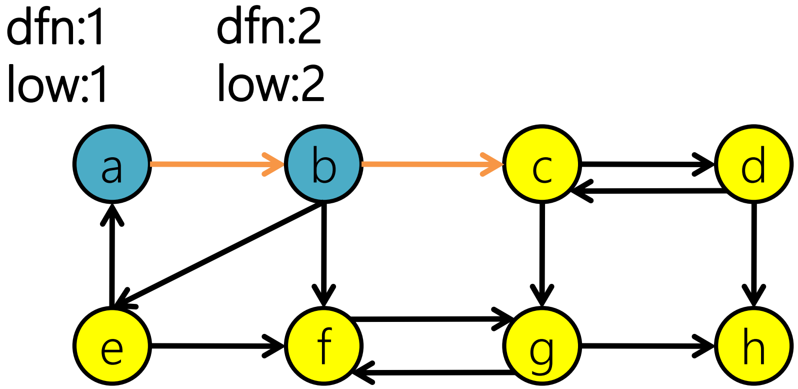
dfn:2
low:2

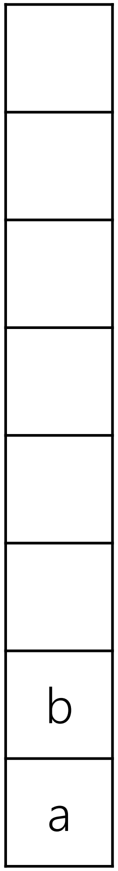

b

a

SCC

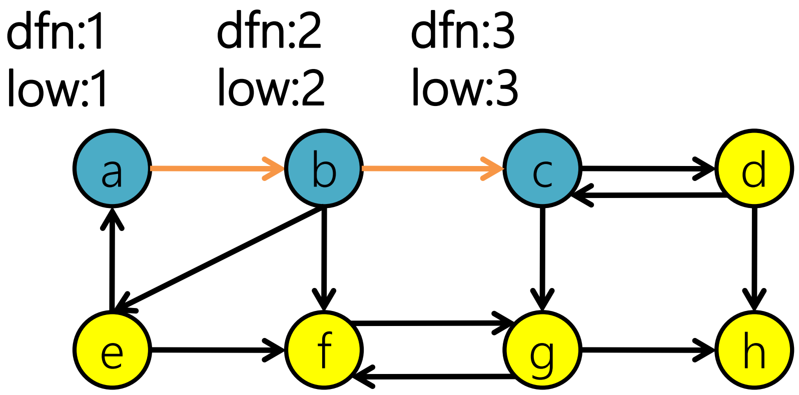
- Tarjan

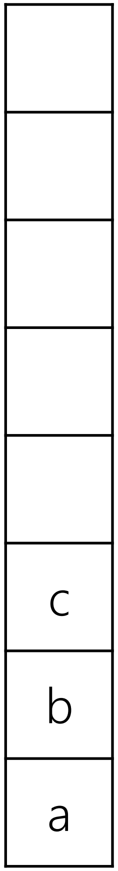




SCC

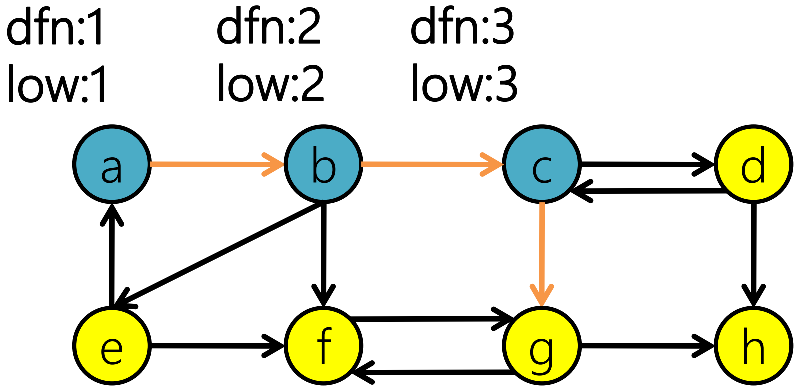
- Tarjan

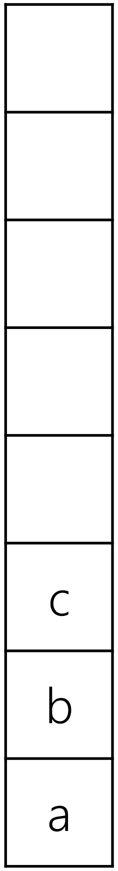




SCC

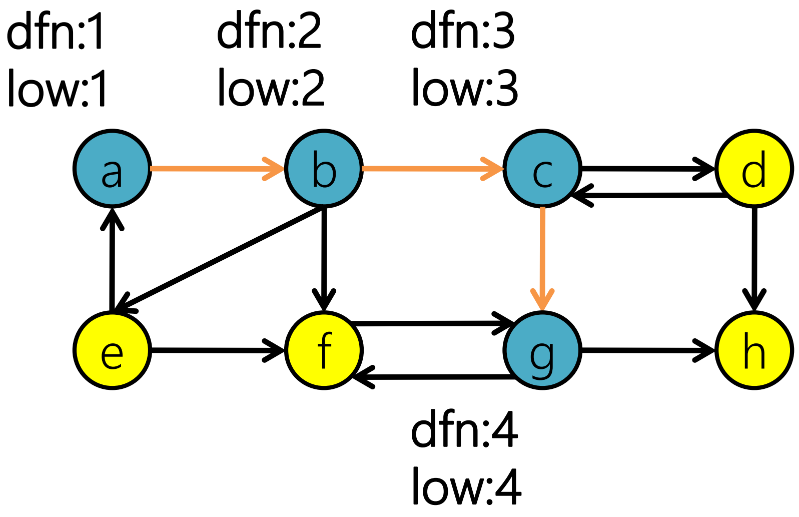
- Tarjan





SCC

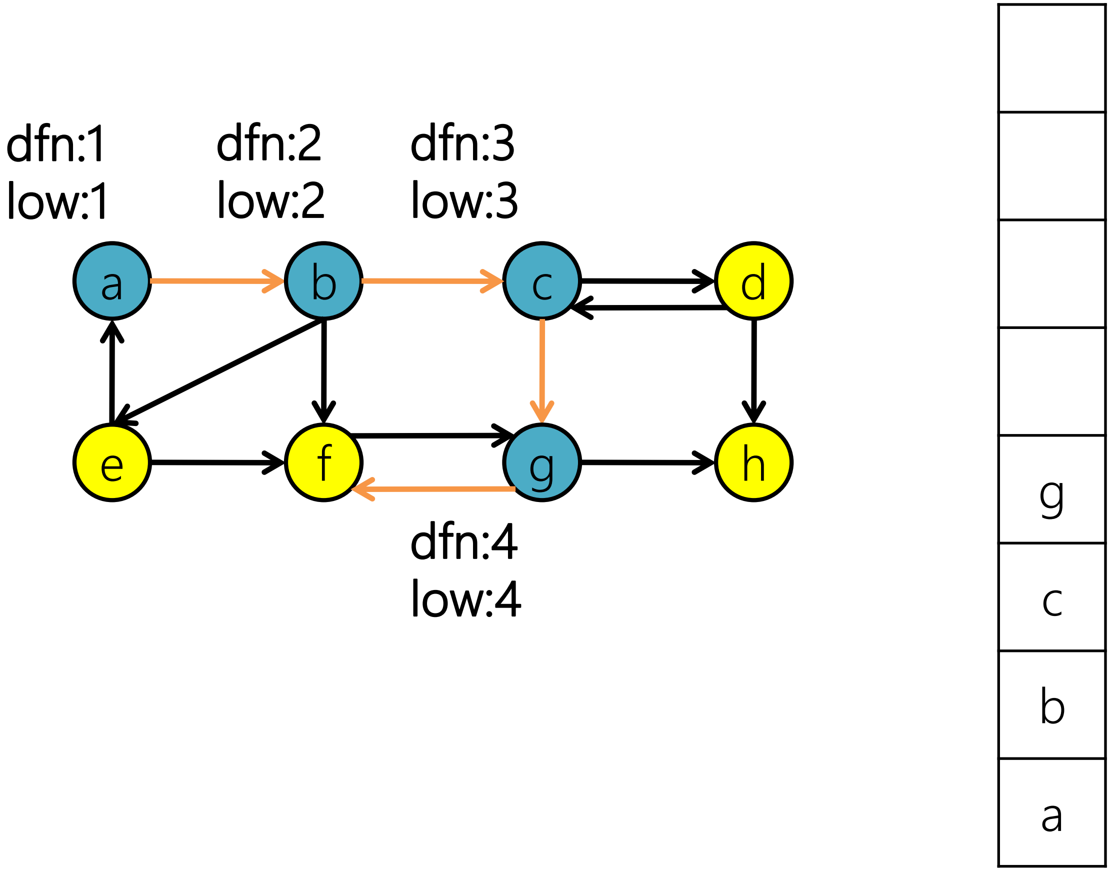
- Tarjan



g
c
b
a

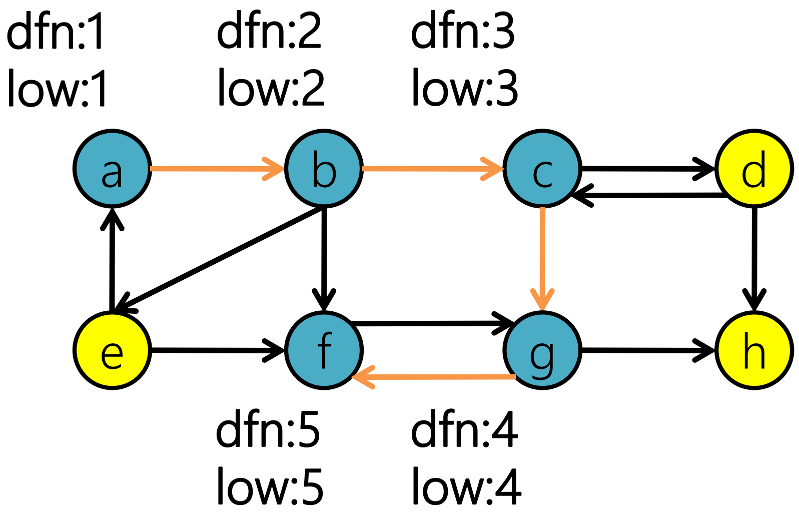
SCC

- Tarjan



SCC

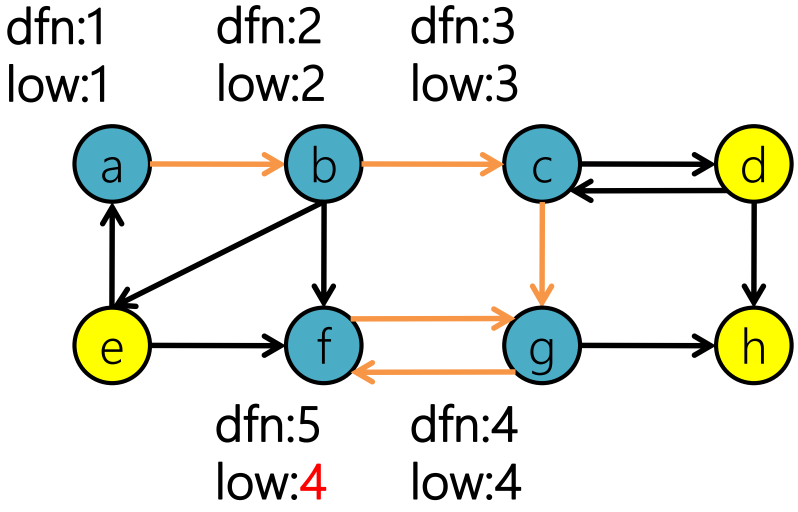
- Tarjan



f
g
c
b
a

SCC

- Tarjan

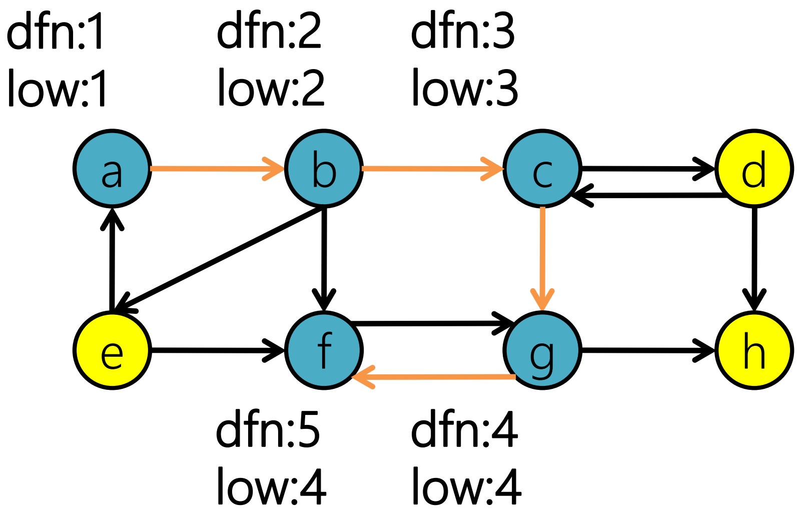


In stack

f
g
c
b
a

SCC

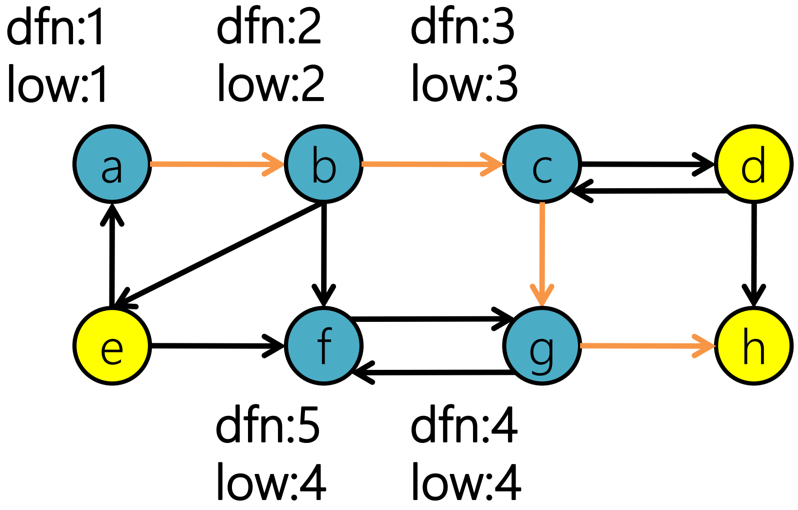
- Tarjan



f
g
c
b
a

SCC

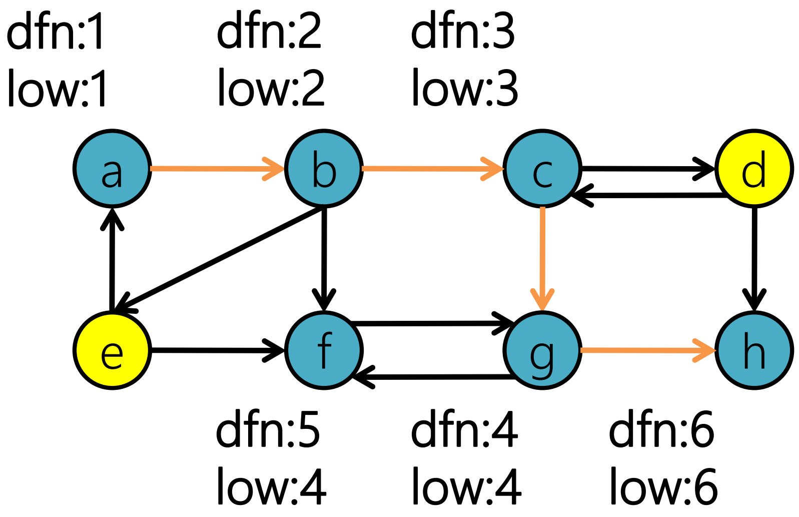
- Tarjan



f
g
c
b
a

SCC

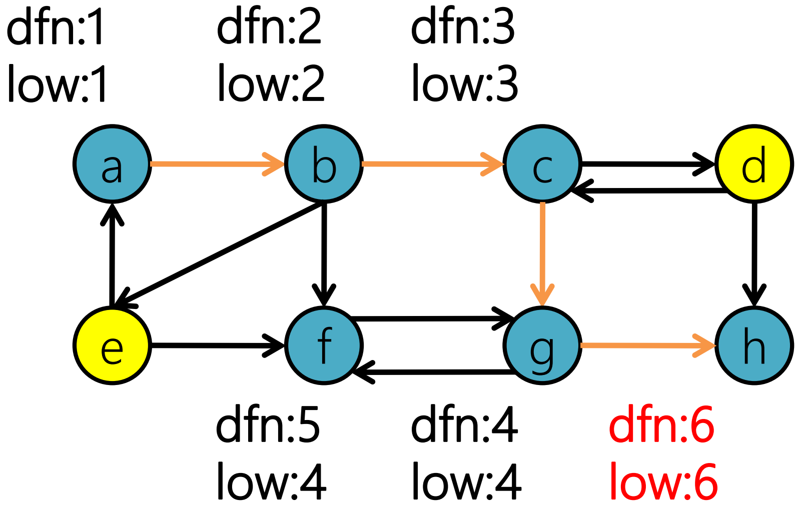
- Tarjan



h
f
g
c
b
a

SCC

- Tarjan

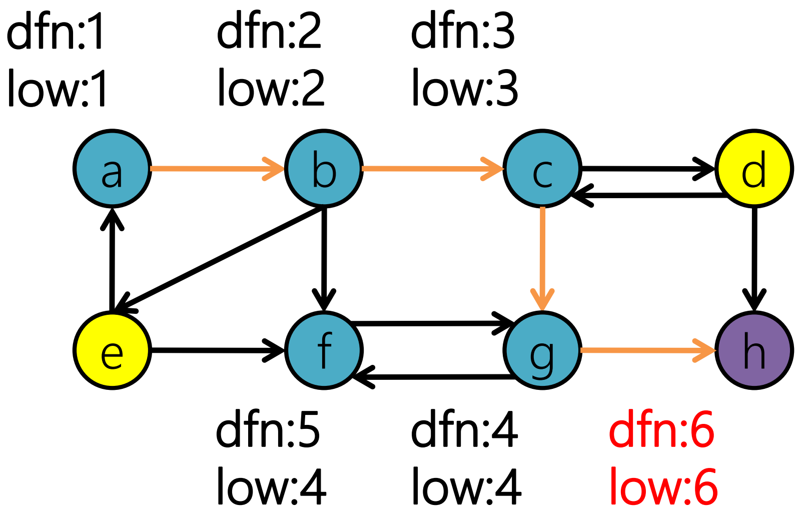


dfn == low

h
f
g
c
b
a

SCC

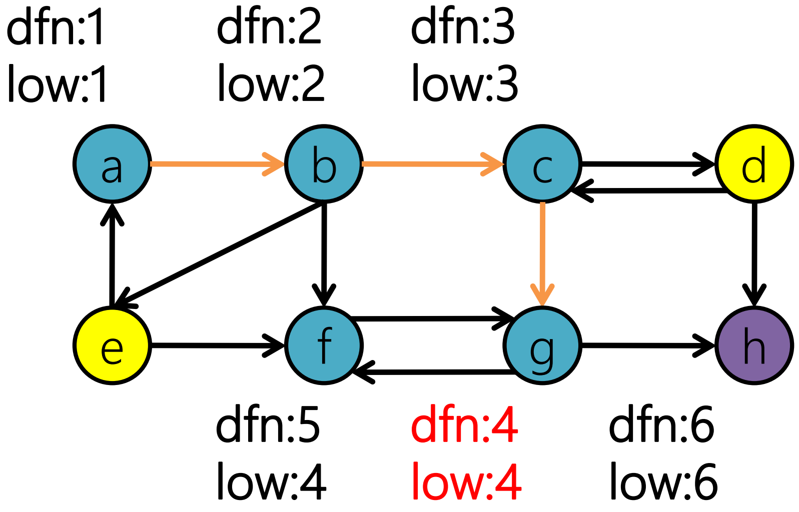
- Tarjan



f
g
c
b
a

SCC

- Tarjan

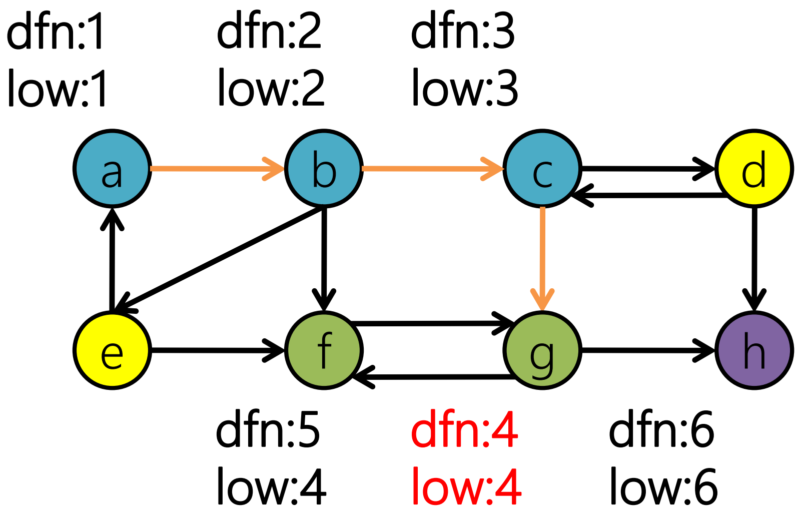


f
g
c
b
a

dfn == low

SCC

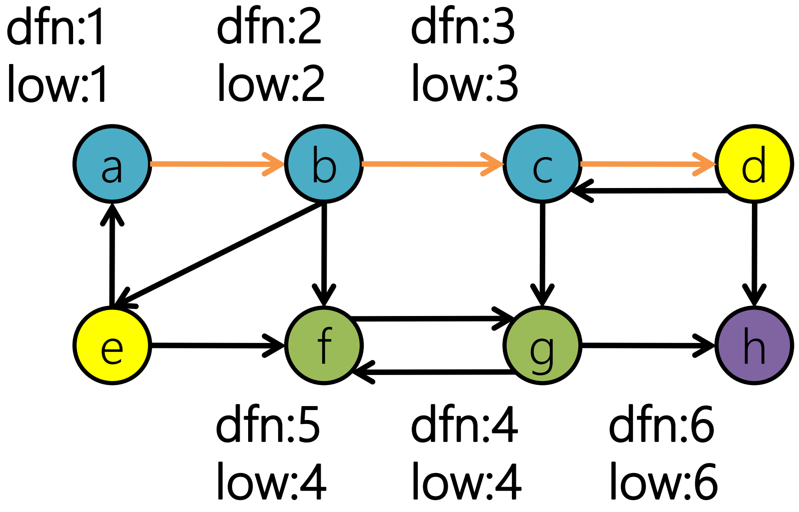
- Tarjan



c
b
a

SCC

- Tarjan



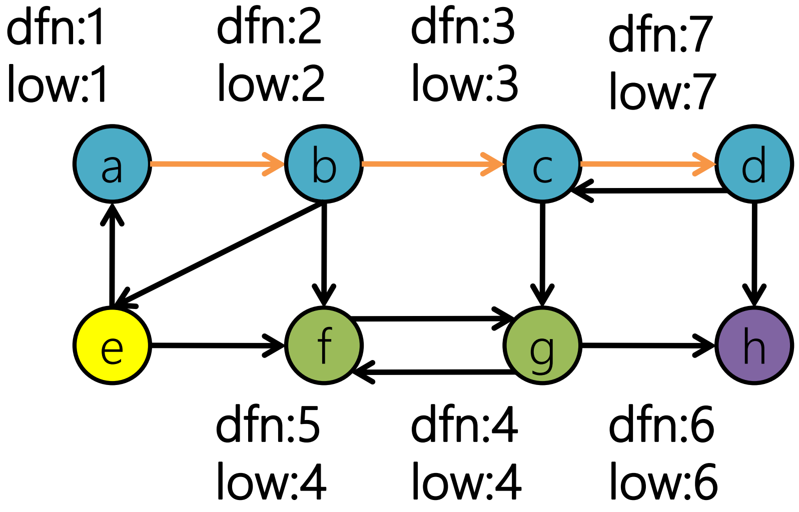
c

b

a

SCC

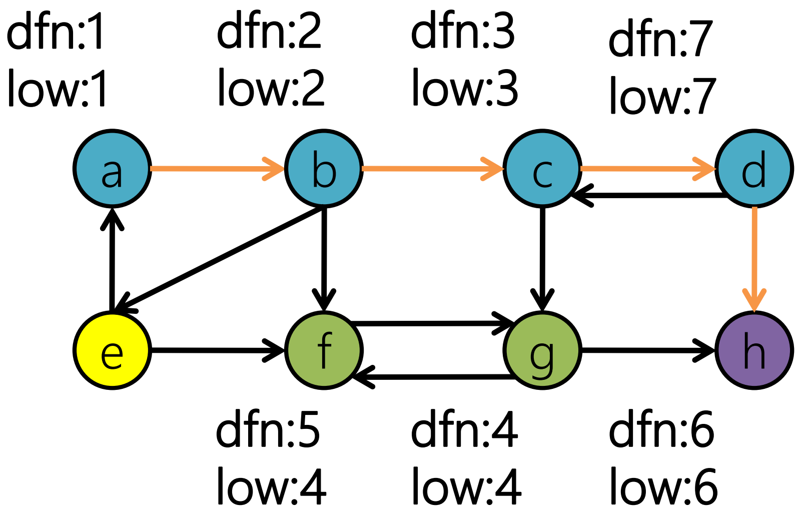
- Tarjan

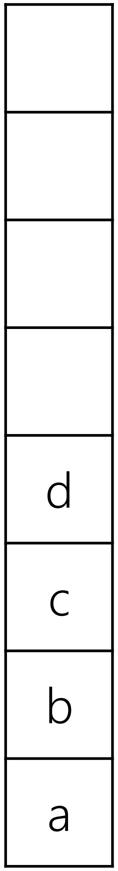


d
c
b
a

SCC

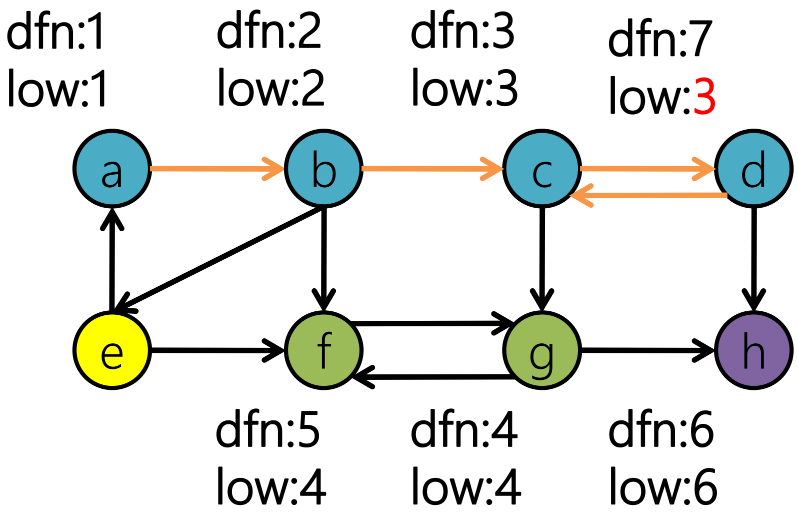
- Tarjan


dfn:1
low:1
dfn:2
low:2
dfn:3
low:3
dfn:7
low:7
a
b
c
d
e
f
g
h
dfn:5
low:4
dfn:4
low:4
dfn:6
low:6


d
c
b
a

SCC

- Tarjan



In stack

d

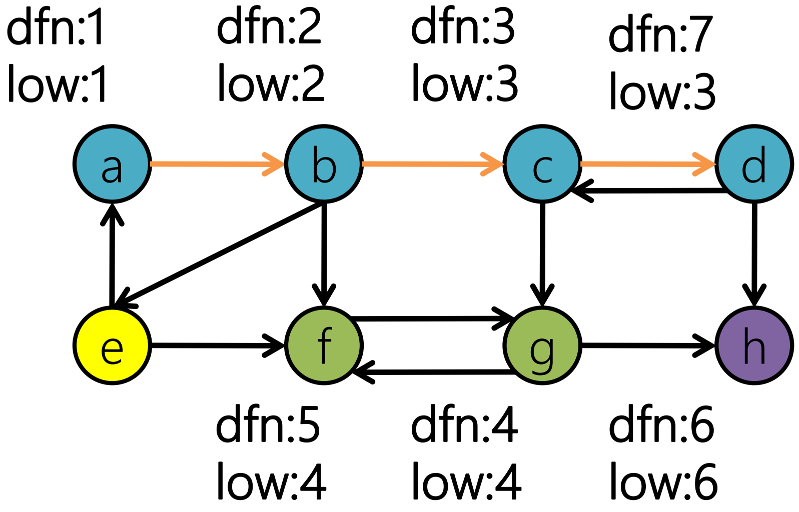
c

b

a

SCC

- Tarjan



d

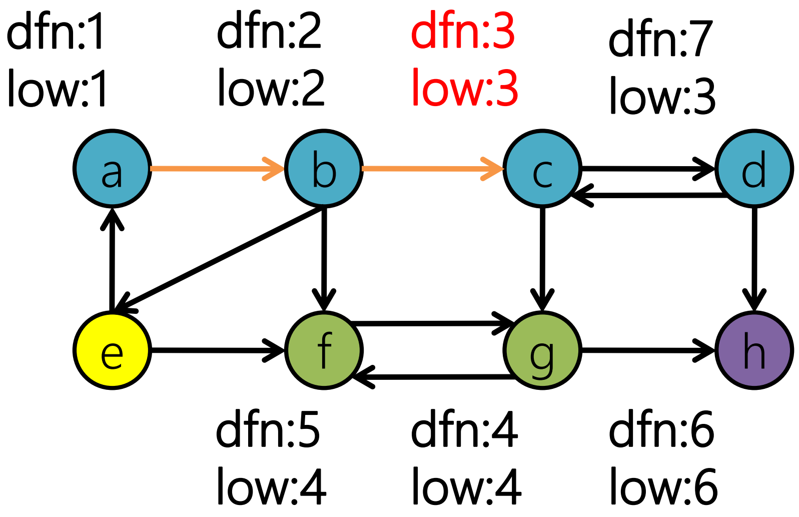
c

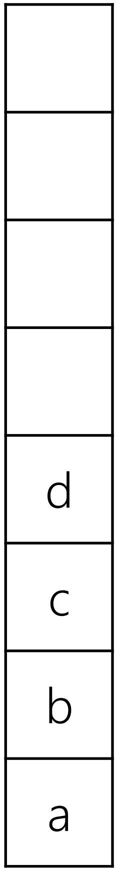
b

a

SCC

- Tarjan

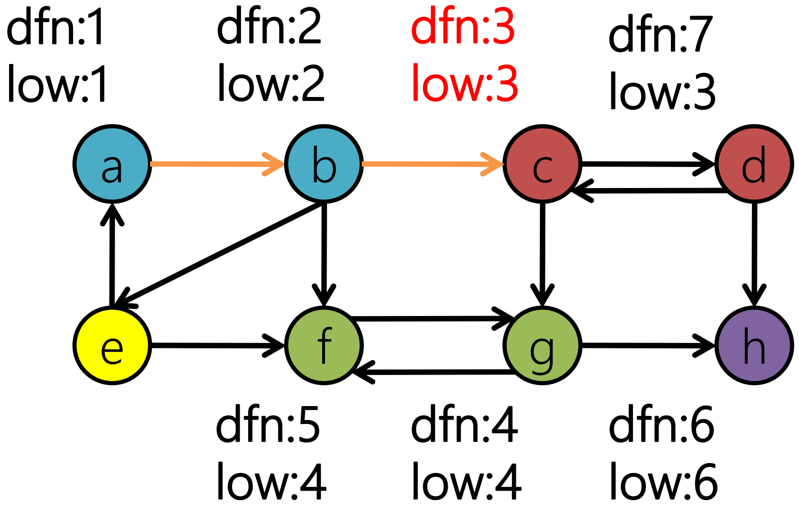


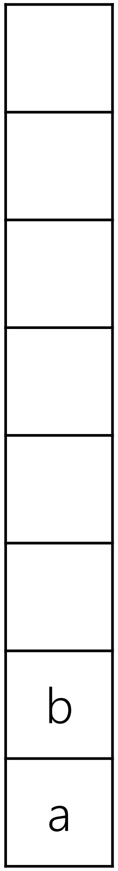


dfn == low

SCC

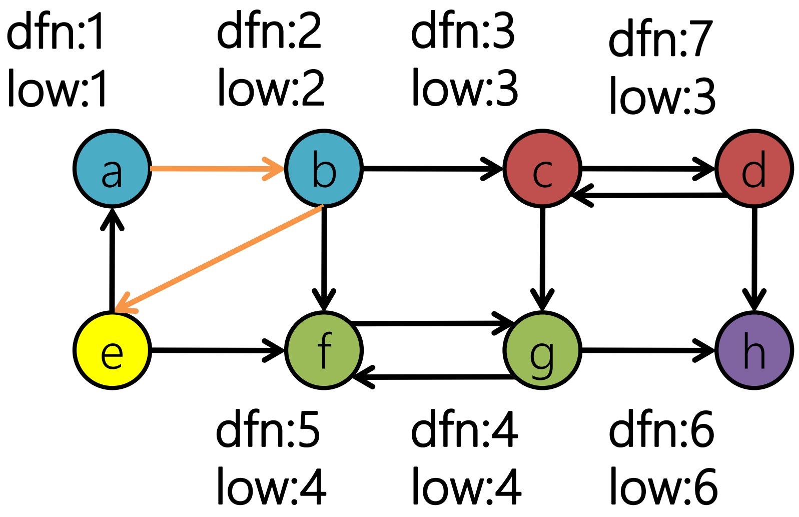
- Tarjan


dfn:1
low:1
dfn:2
low:2
dfn:3
low:3
dfn:7
low:3
a
b
c
d
e
f
g
h
dfn:5
low:4
dfn:4
low:4
dfn:6
low:6


b
a

SCC

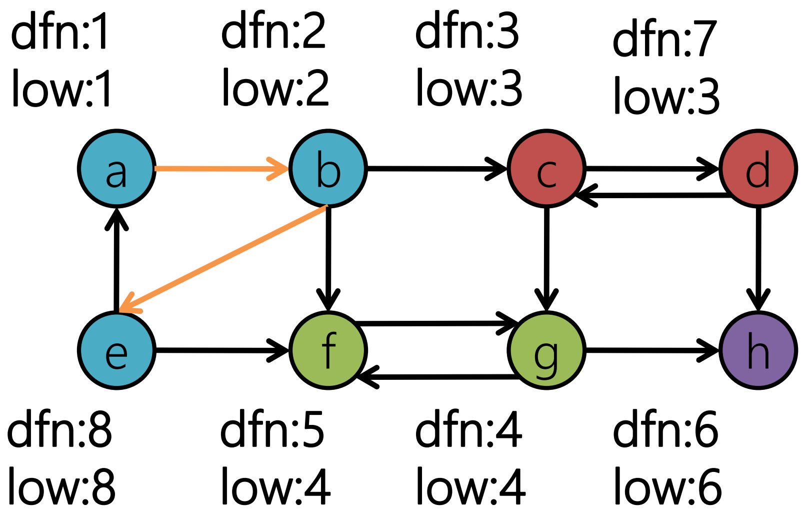
- Tarjan

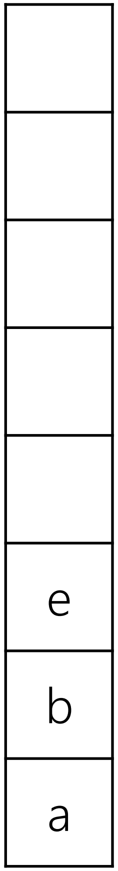


b
a

SCC

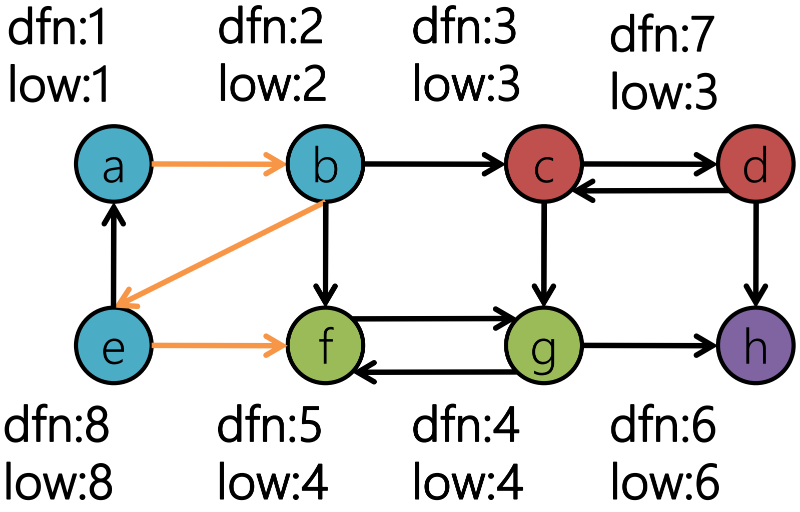
- Tarjan


dfn:1
low:1
dfn:2
low:2
dfn:3
low:3
dfn:7
low:3
a
b
c
d
e
f
g
h
dfn:8
low:8
dfn:5
low:4
dfn:4
low:4
dfn:6
low:6


e
b
a

SCC

- Tarjan



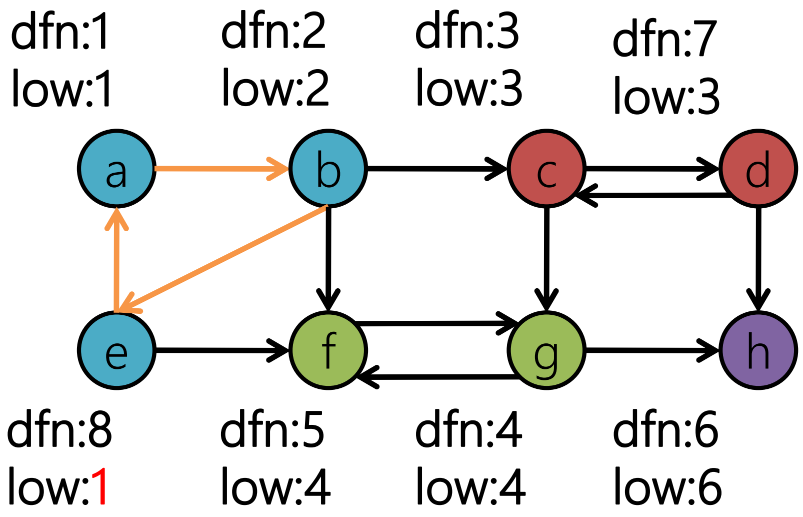
e

b

a

SCC

- Tarjan



dfn:1
low:1

dfn:2
low:2

dfn:3
low:3

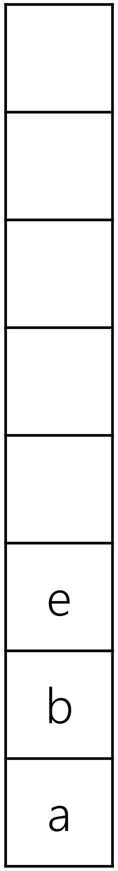
dfn:7
low:3

dfn:8
low:1

dfn:5
low:4

dfn:4
low:4

dfn:6
low:6



e

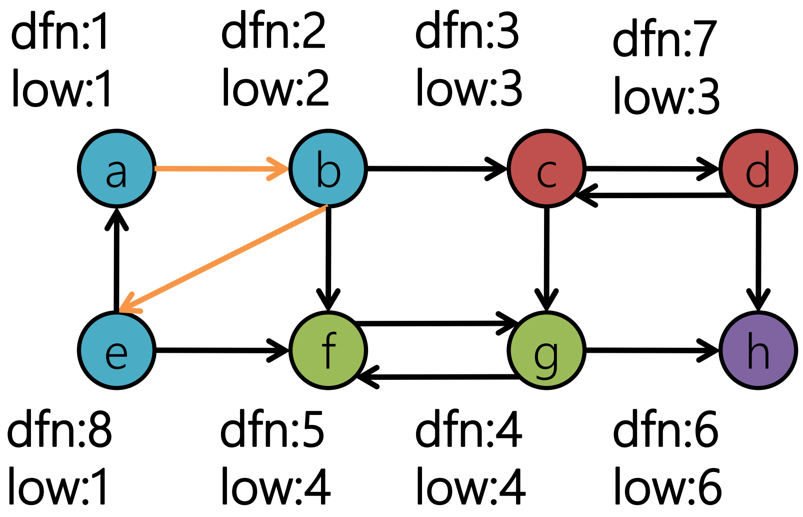
b

a

In stack

SCC

- Tarjan



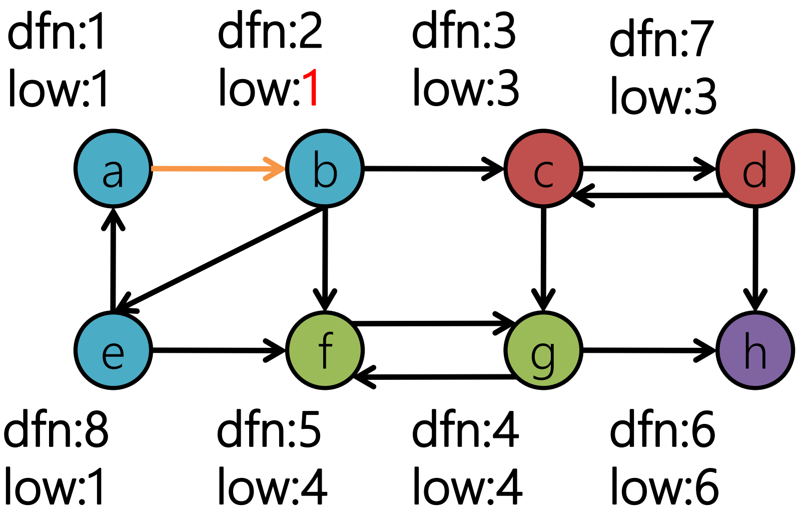
e

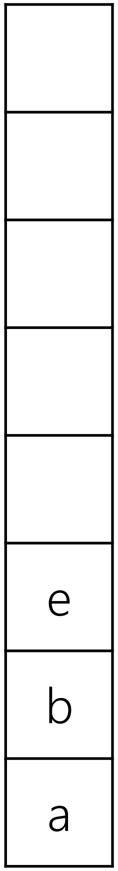
b

a

SCC

- Tarjan



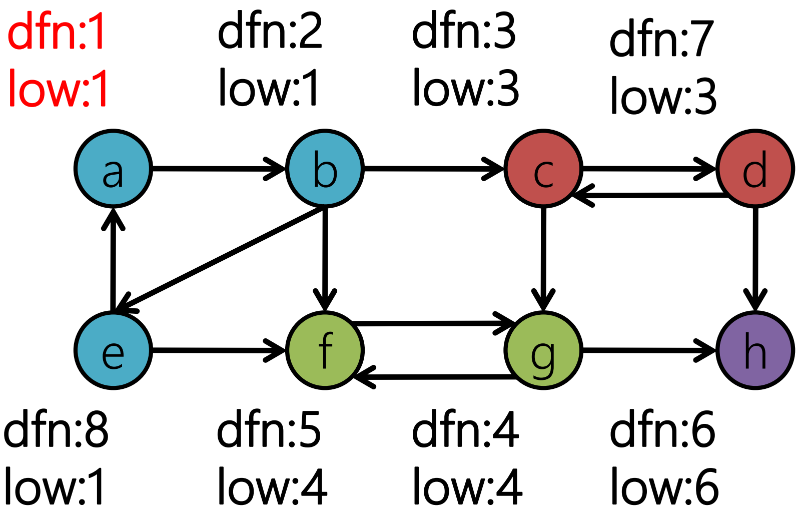


children's low < low

made by riljian

SCC

- Tarjan



dfn:1
low:1

dfn:2
low:1

dfn:3
low:3

dfn:7
low:3

dfn:8
low:1

dfn:5
low:4

dfn:4
low:4

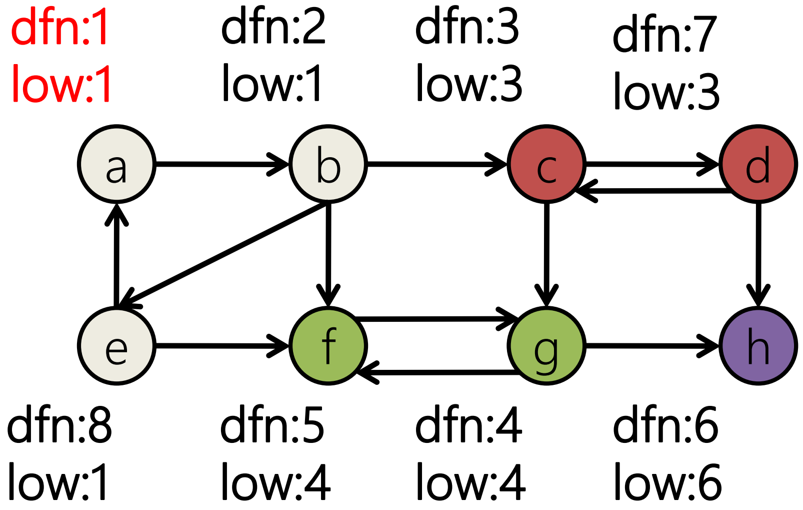
dfn:6
low:6

dfn == low

e
b
a

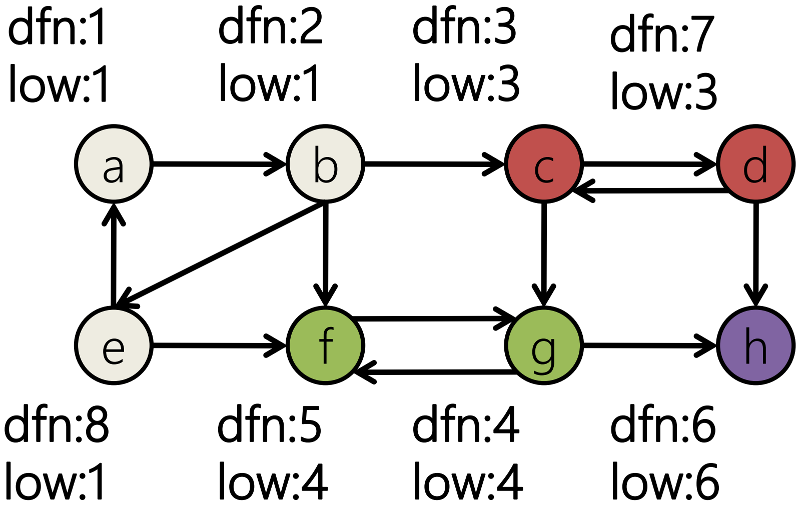
SCC

- Tarjan


dfn:1
low:1
dfn:2
low:1
dfn:3
low:3
dfn:7
low:3
a
b
c
d
e
f
g
h
dfn:8
low:1
dfn:5
low:4
dfn:4
low:4
dfn:6
low:6

SCC

- Tarjan



4 strongly connected components

Practice

Uva - 11838

Practice

UVa 11504