NCKU Programming Contest Training Course
2013/07/17

Pin-chieh Huang (free999)

pinchieh.huang@gmail.com

http://myweb.ncku.edu.tw/~p76014143/20130717.rar

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

made by free999 & electron

Outline

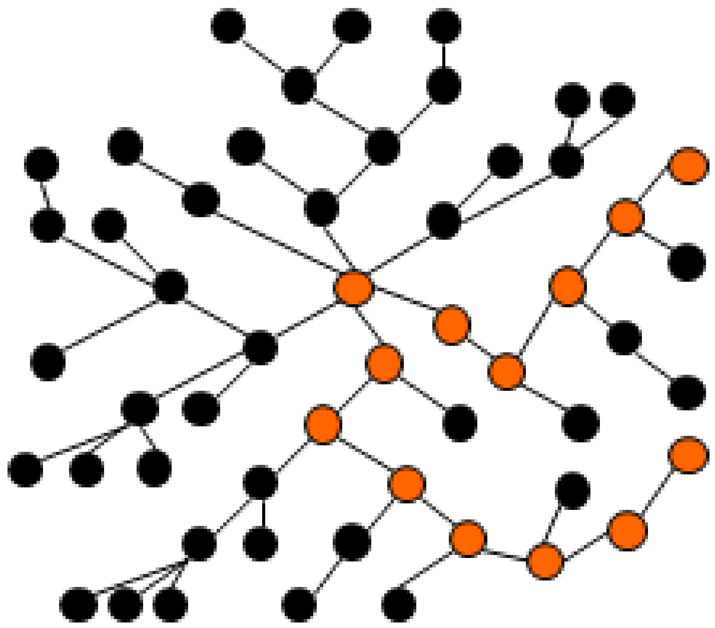
Tree - diameter

Cut vertex and Cut edge

Strongly-connected component

Diameter

- 一棵無根樹的「直徑」，就是相離最遠的兩個點的距離。



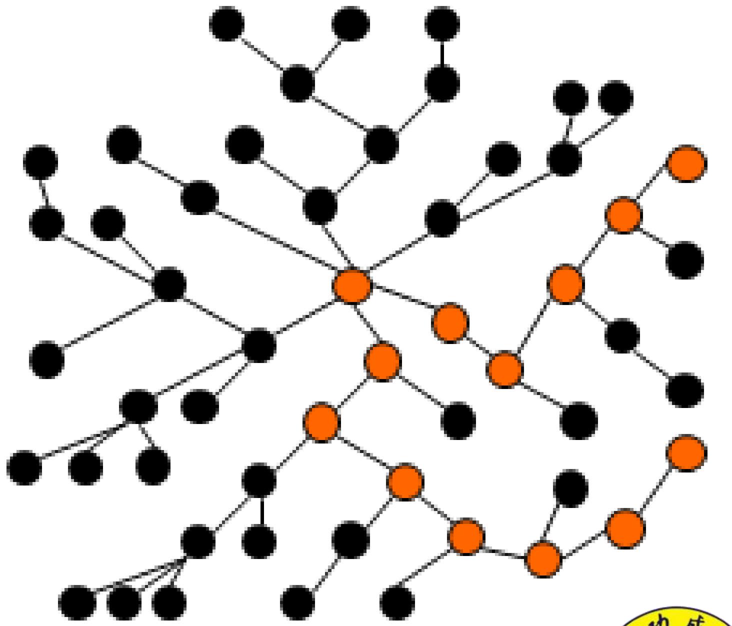
Diameter

- 任選一樹根

Two method

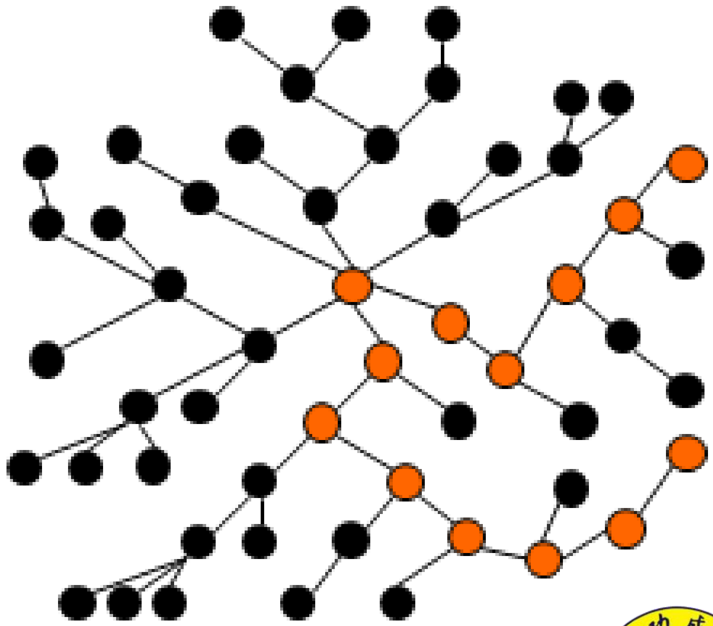
1. 找出最長和次長的DFS路徑
2. 找出最長的DFS路徑終點 V,

以 V 為 root 找出最長的DFS路徑



Example

Uva 10308



Outline

Tree - diameter

↓

Cut vertex and Cut edge

↓

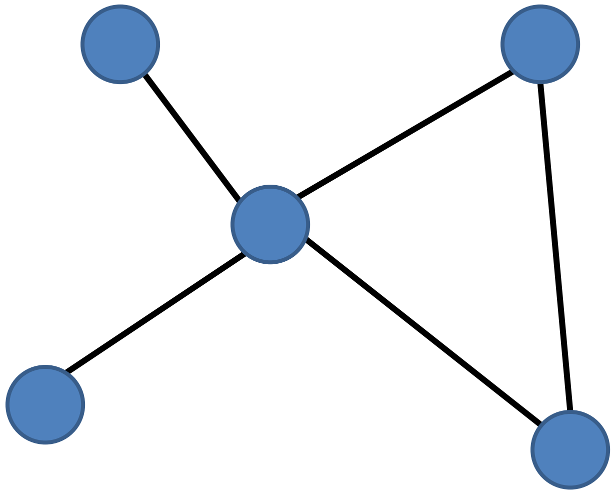
Strongly-connected component

Cut Vertex and Edge

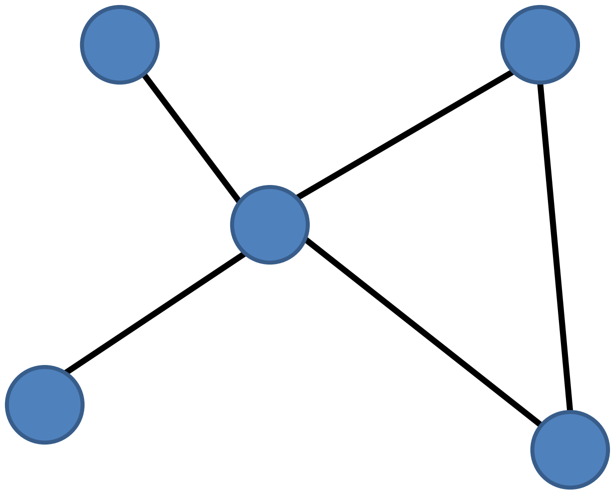
- Cut Vertex
  - In mathematics and computer science, a **cut vertex** or **articulation point** is a vertex of a graph such that removal of the vertex causes an increase in the number of connected components. If the graph was connected before the removal of the vertex, it will be disconnected afterwards. Any connected graph with a cut vertex has a connectivity of 1.
  - While well-defined even for directed graphs, cut vertices are primarily used in undirected graphs. In general, a connected, undirected graph with *n* vertices can have no more than *n*-2 cut vertices. Naturally, a graph may have no cut vertices at all.

Cut Vertex

- Cut Vertex

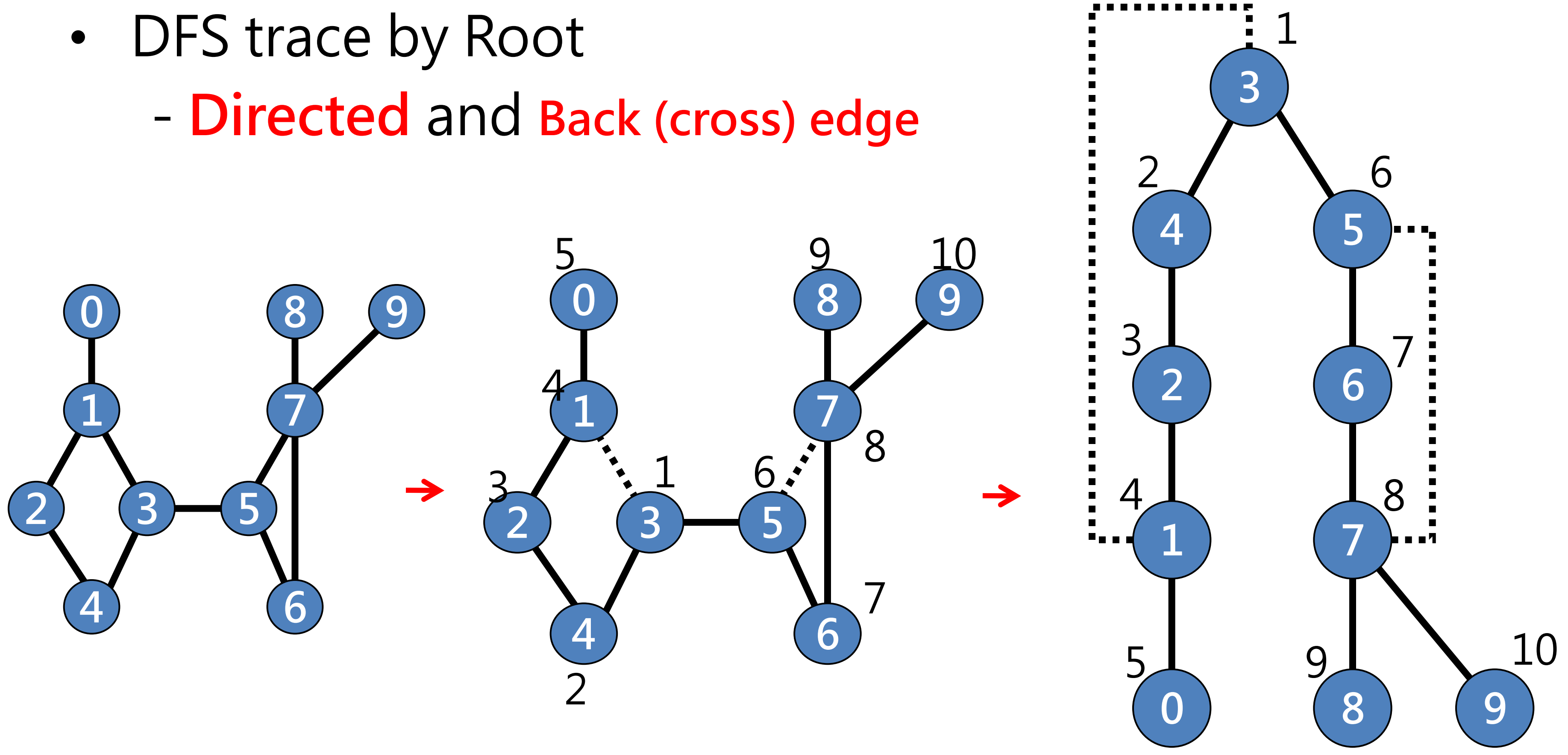


and



Cut Vertex

- DFS trace by Root
  - Directed and Back (cross) edge

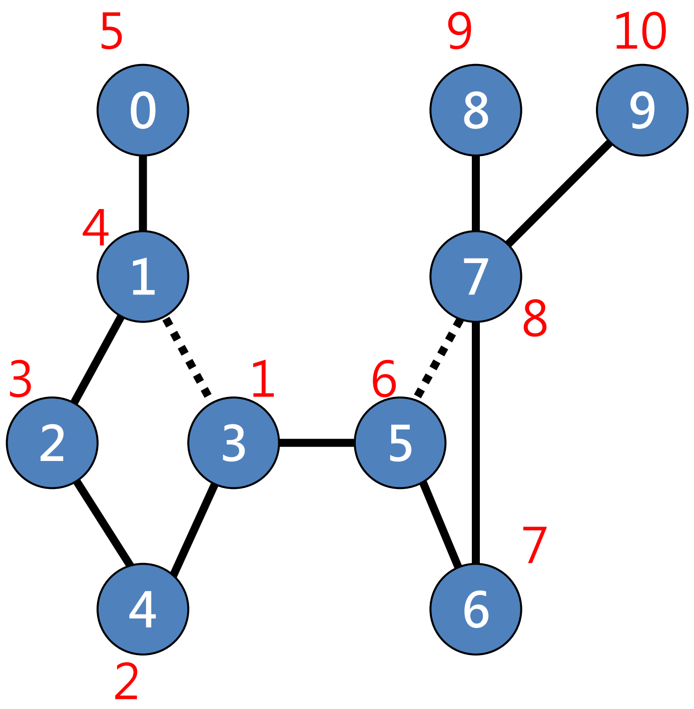


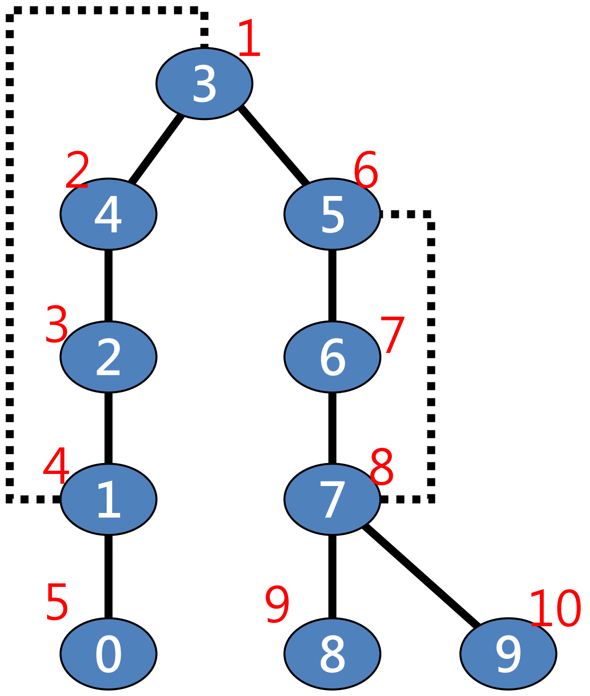
Cut Vertex

- Observation
  - If root has two child, => the root is an articulation point
  - If a vertex u has a child w, so that w and w can' t back to u' s parent => u is articulation point
- Define
  - low(u): the minimum dfn value can obtain by u
  - low(u)
    - (1) **Directed Edge:** min{ dfn(u), min{low(w)|w是u的child},
    - (2) **Back (cross) edge:** min{dfn(w)|(u,w)是back edge}}

Cut Vertex

Vertex	0	1	2	3	4	5	6	7	8	9
dfn	5	4	3	1	2	6	7	8	9	10
low	5	1	1	1	1	6	6	6	9	10

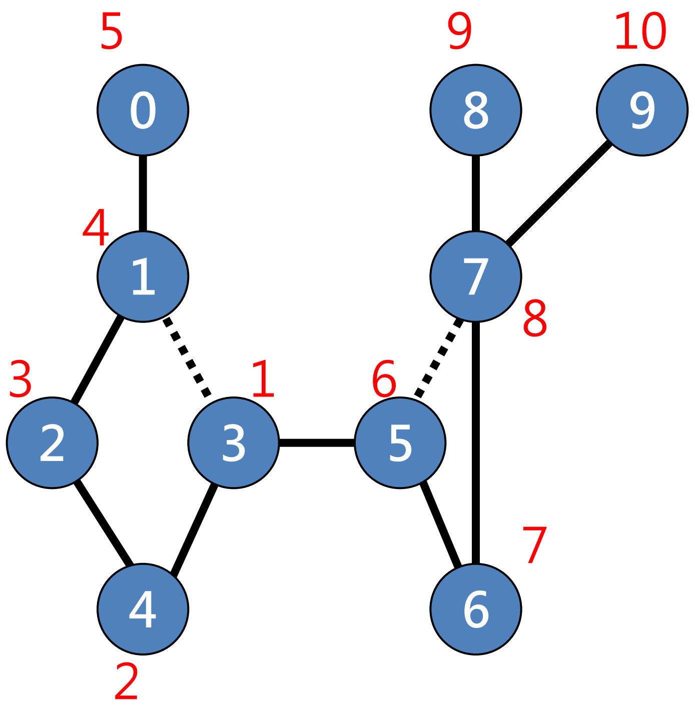




made by free999 & electron

Cut Vertex

Vertex	0	1	2	3	4	5	6	7	8	9
dfn	5	4	3	1	2	6	7	8	9	10
low	5	1	1	1	1	6	6	6	9	10

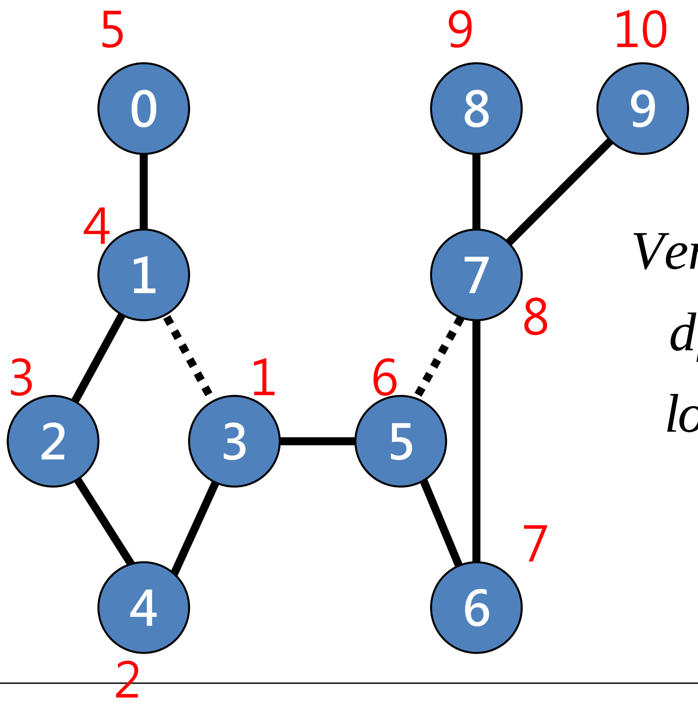


```
dfn[u]=low[u]=deapth++;
for(w=0;w<MAX_VERTEX;w++)
    if(graph[u][w])
    {
        if(dfn[w]<0)     //w isn't visited.
        {
            dfnlow(w,u);
            child++;
            if(dfn[u]<=low[w])yes=1;
            low[u]=(low[u]<low[w])?low[u]:low[w];
        }
        else if(w!=v)  //Back edge
            low[u]=(low[u]<dfn[w])?low[u]:dfn[w];
    }
if((child>1||v>=0)&&yes)answer[ansc++]=u;
```

made by free999 & electron

Cut Edge

- Observation
  - Similar to cut vertex
  - Given an edge (u, v), if now is tracing u -> v and low[v] > dfn[u] => cut edge



Vertex	0	1	2	3	4	5	6	7	8	9
dfn	5	4	3	1	2	6	7	8	9	10
low	5	1	1	1	1	6	6	6	9	10

made by free999 & electron

Example

- Uva 315

Outline

Tree - diameter

$\downarrow$

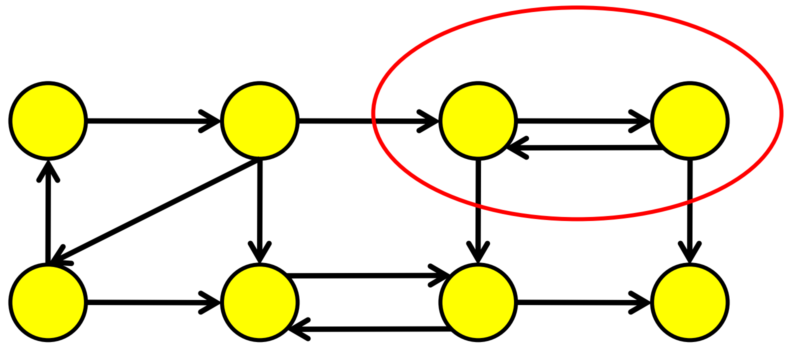
Cut vertex and Cut edge

$\downarrow$

Strongly-connected component

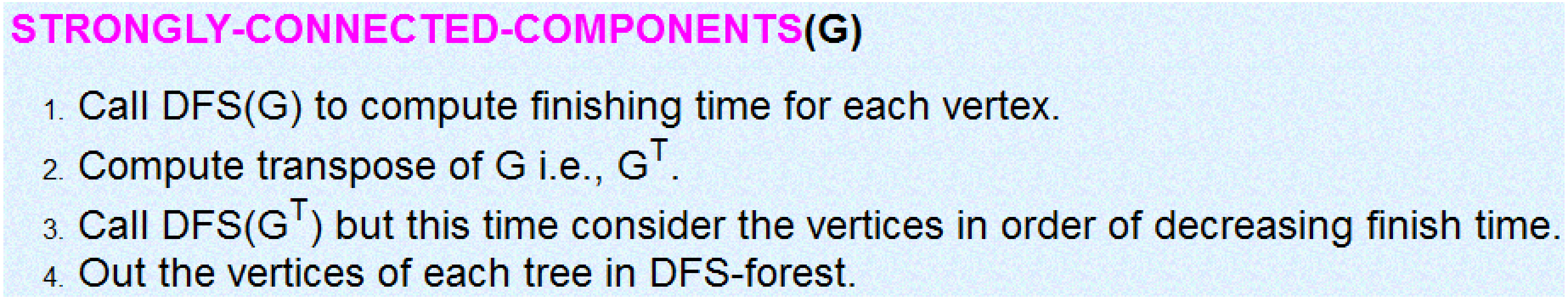
SCC

- SCC
  - Strongly-connected component



SCC

- Algorithm

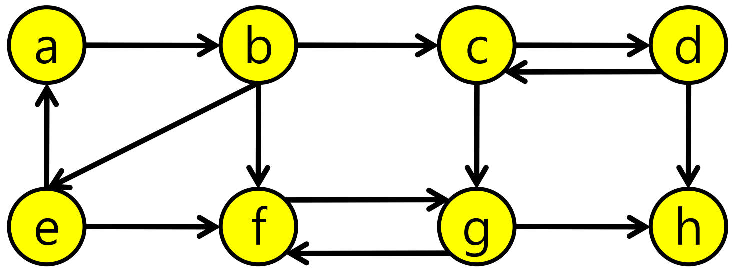


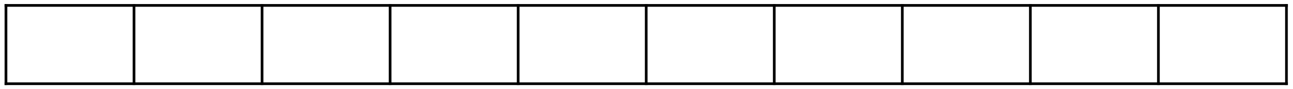
```
STRONGLY-CONNECTED-COMPONENTS(G)

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.
```

SCC

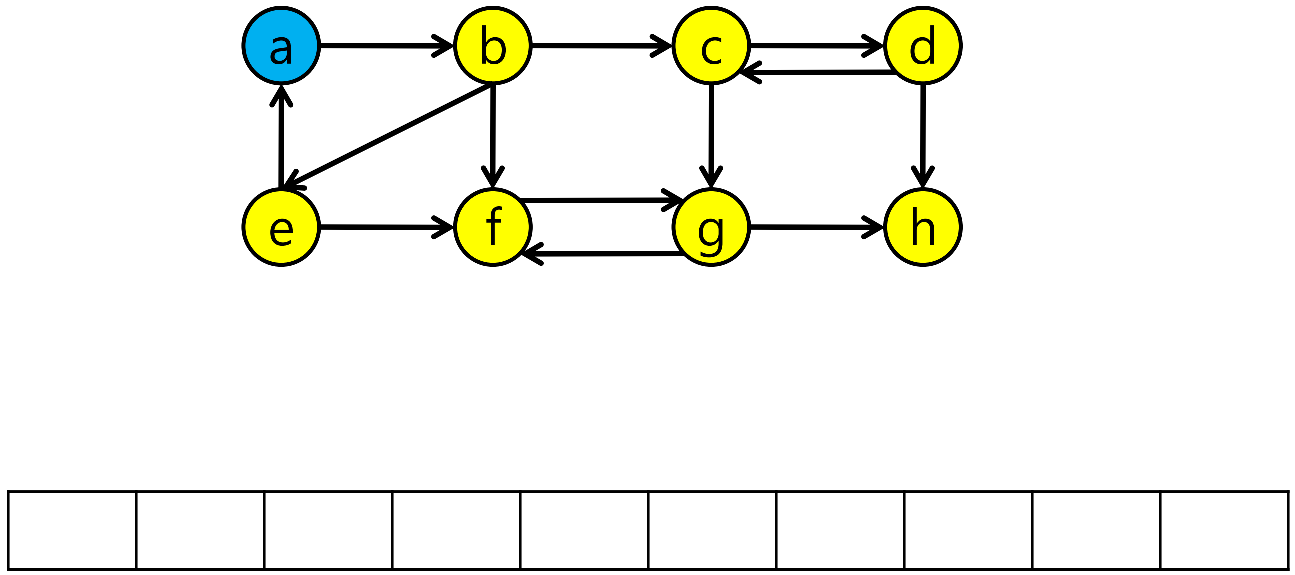
- Algorithm





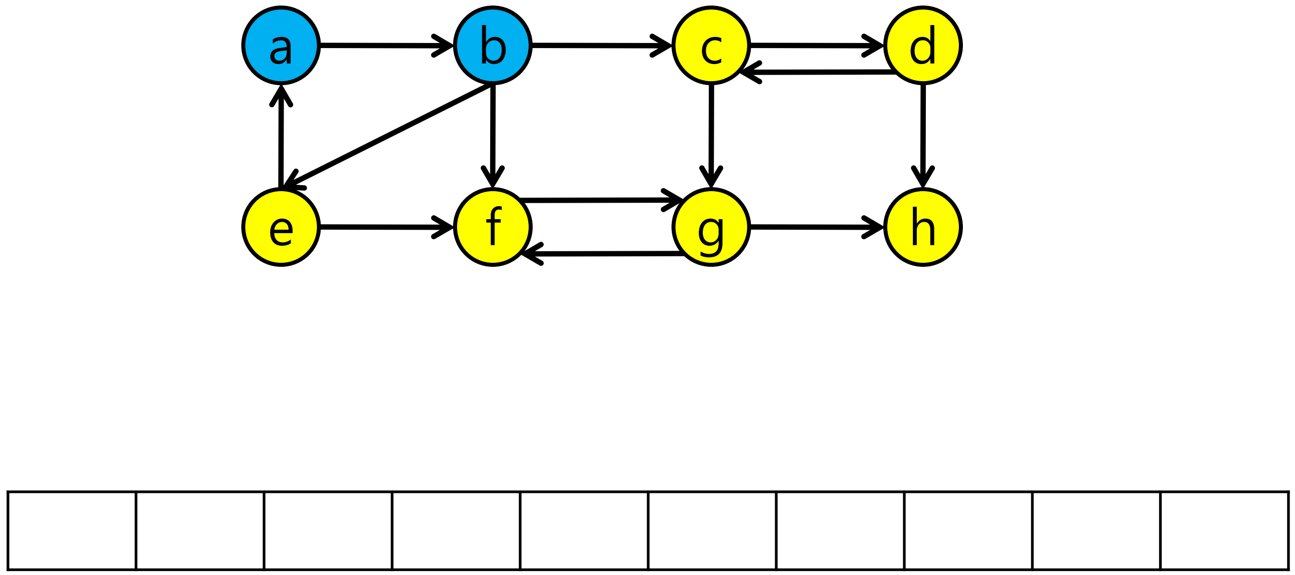
SCC

- Algorithm



SCC

- Algorithm



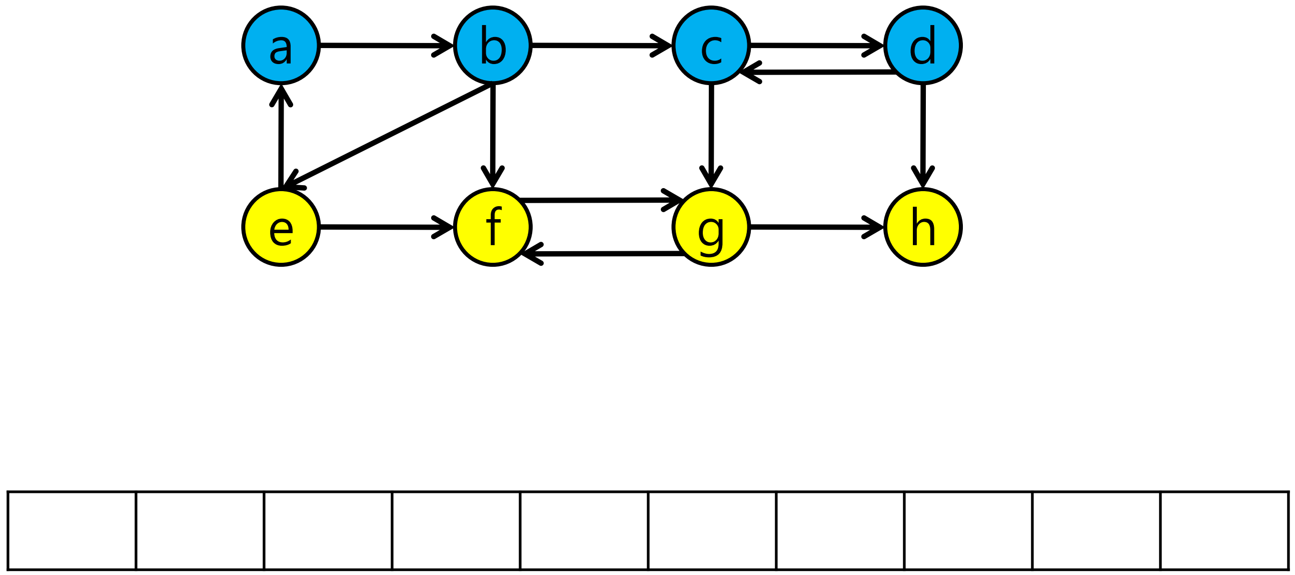
SCC

- Algorithm

$$\begin{tikzcd}
a \arrow[r] & b \arrow[r] \arrow[dl] \arrow[d] & c \arrow[r, shift left] \arrow[d] & d \arrow[l, shift left] \arrow[d] \\
e \arrow[u] \arrow[r] & f \arrow[r, shift left] & g \arrow[l, shift left] \arrow[r] & h
\end{tikzcd}$$

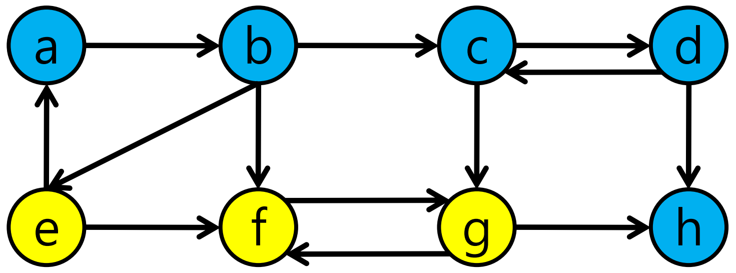
SCC

- Algorithm



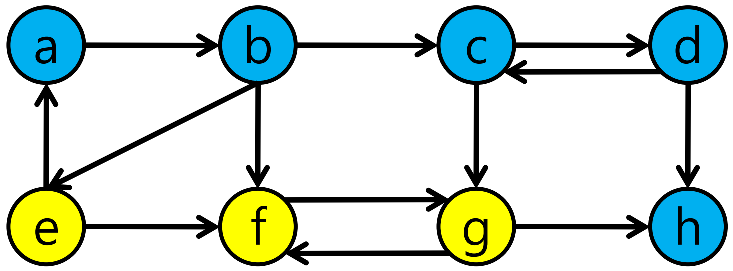
SCC

- Algorithm



SCC

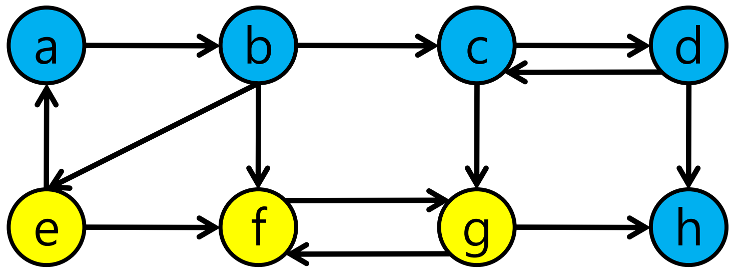
- Algorithm



h									

SCC

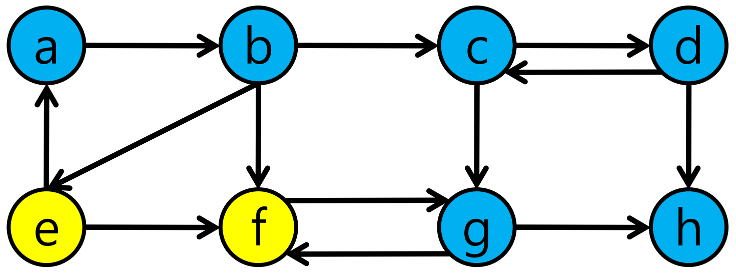
- Algorithm



h	d								

SCC

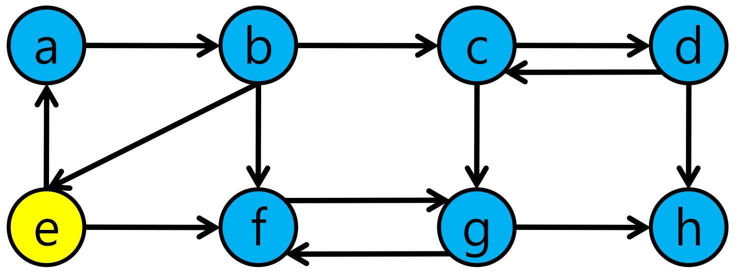
- Algorithm



h	d								

SCC

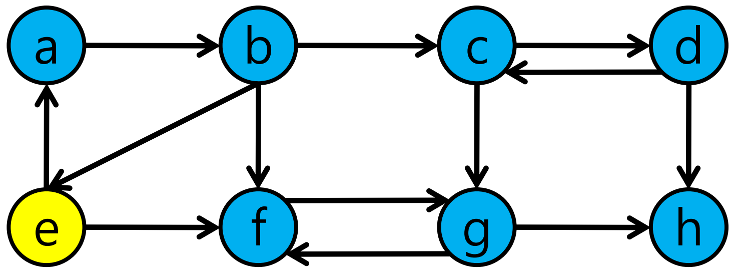
- Algorithm



h	d								

SCC

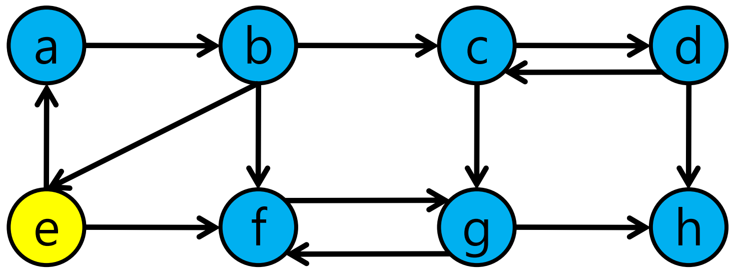
- Algorithm



h	d	f							

SCC

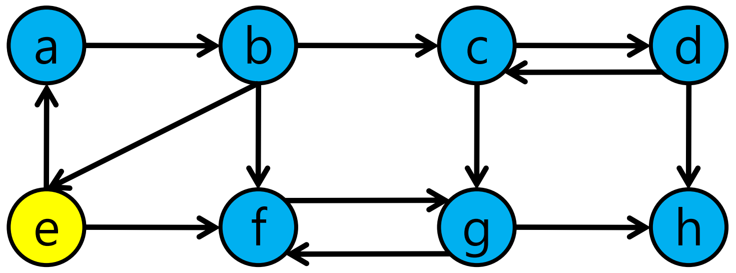
- Algorithm



h	d	f	g						

SCC

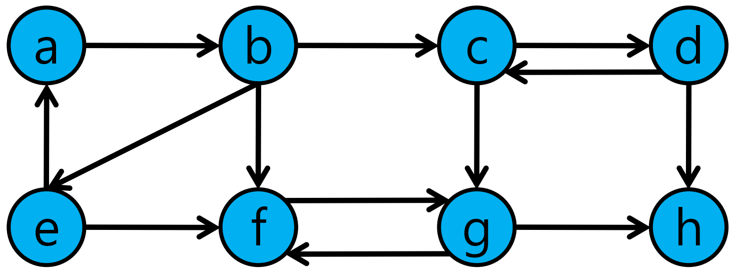
- Algorithm



h	d	f	g	c					

SCC

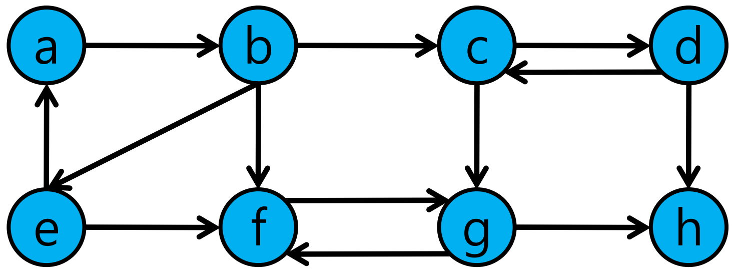
- Algorithm



h	d	f	g	c					

SCC

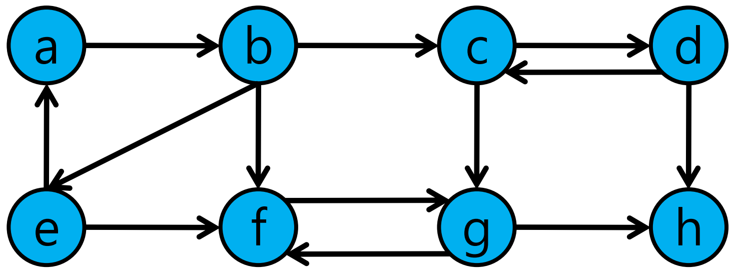
- Algorithm



h	d	f	g	c	e				

SCC

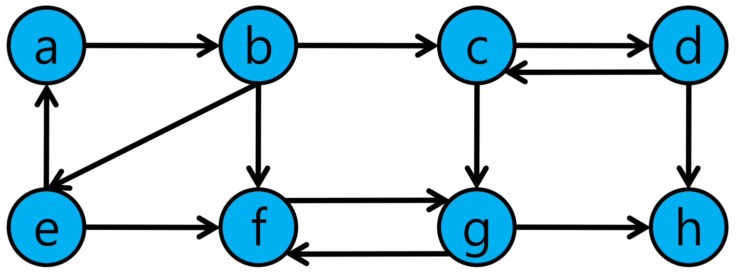
- Algorithm



h	d	f	g	c	e	b			

SCC

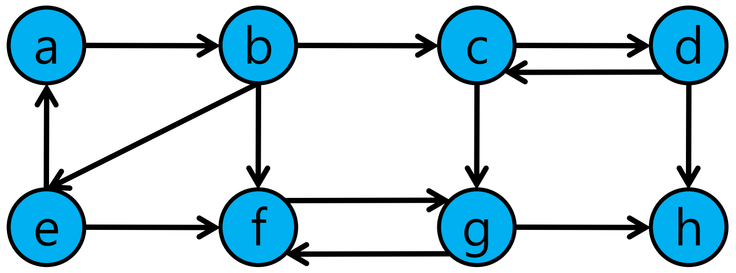
- Algorithm



h	d	f	g	c	e	b	a		

SCC

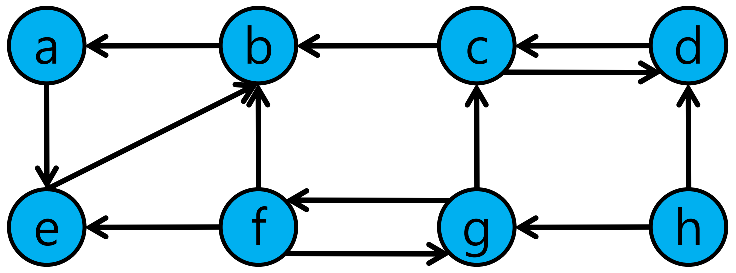
- Algorithm
  - Reverse the graph



h	d	f	g	c	e	b	a		

SCC

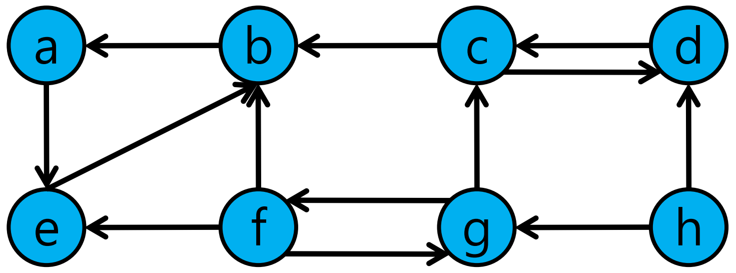
- Algorithm
  - Reverse the graph



h	d	f	g	c	e	b	a		

SCC

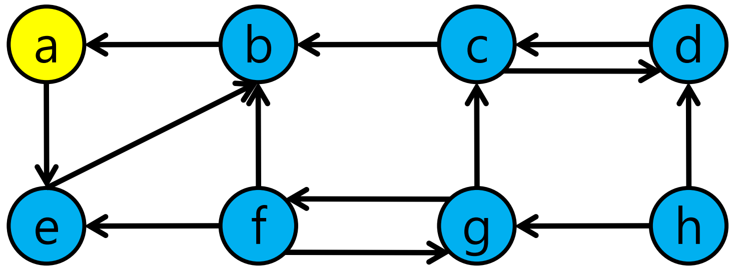
- Algorithm
  - Reverse the graph
  - Re-search by the ending time



h	d	f	g	c	e	b	a		

SCC

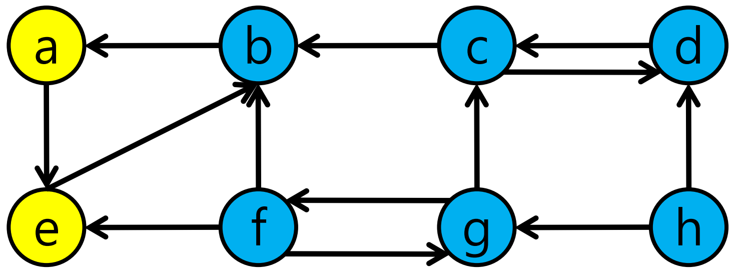
- Algorithm
  - Reverse the graph
  - Re-search by the ending time



h	d	f	g	c	e	b	a		

SCC

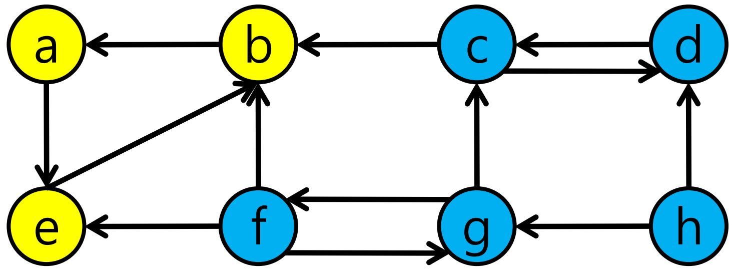
- Algorithm
  - Reverse the graph
  - Re-search by the ending time



h	d	f	g	c	e	b	a		

SCC

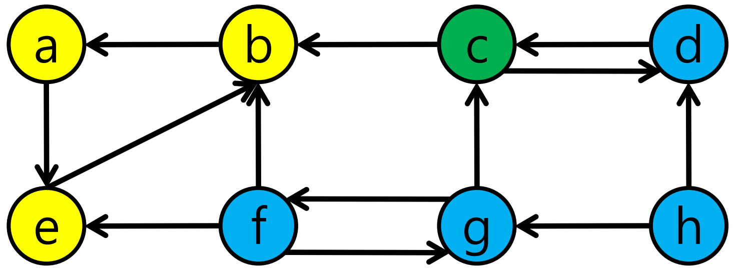
- Algorithm
  - Reverse the graph
  - Re-search by the ending time



h	d	f	g	c	e	b	a		

SCC

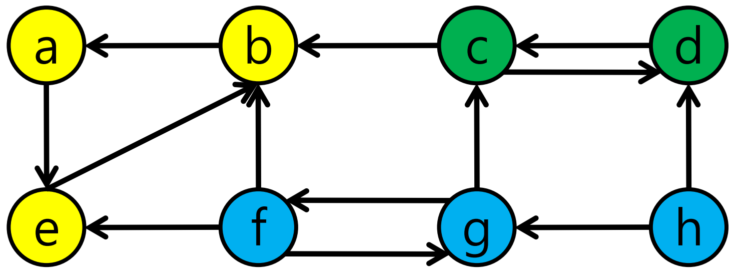
- Algorithm
  - Reverse the graph
  - Re-search by the ending time


$$\begin{tikzcd}
a \arrow[d] & b \arrow[l] & c \arrow[l] \arrow[r, shift right] & d \arrow[l, shift right] \\
e \arrow[ur] & f \arrow[l] \arrow[u] \arrow[r, shift right] & g \arrow[l, shift right] \arrow[u] & h \arrow[l] \arrow[u]
\end{tikzcd}$$

h	d	f	g	c	e	b	a		

SCC

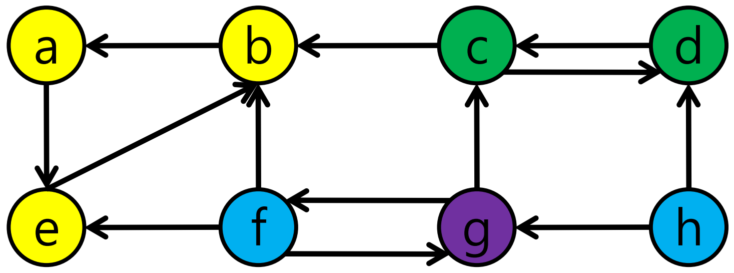
- Algorithm
  - Reverse the graph
  - Re-search by the ending time



h	d	f	g	c	e	b	a		

SCC

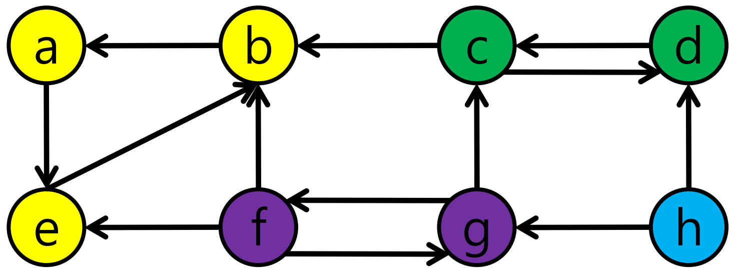
- Algorithm
  - Reverse the graph
  - Re-search by the ending time



h	d	f	g	c	e	b	a		

SCC

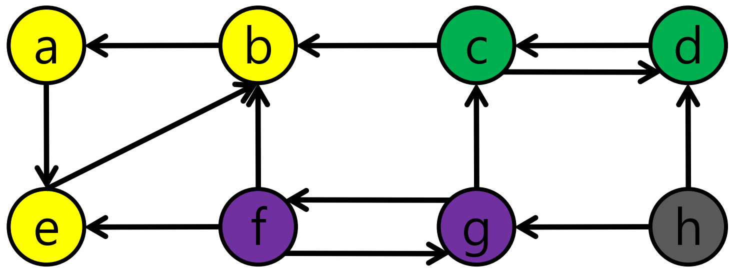
- Algorithm
  - Reverse the graph
  - Re-search by the ending time



h	d	f	g	c	e	b	a		

SCC

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

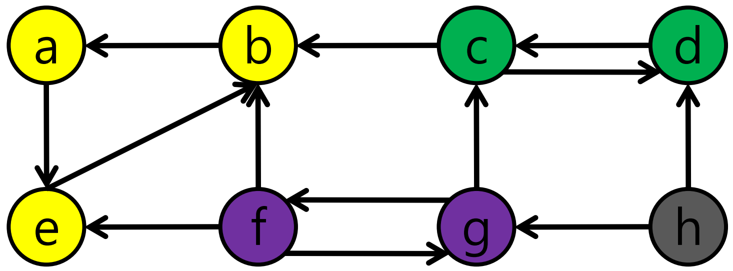


h	d	f	g	c	e	b	a		

SCC

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

4 components



h	d	f	g	c	e	b	a		

Example

- ICPC 4262

http://goo.gl/WctgJ

Homework

- Cut vertex and edge:
  315, 352, 610, 793, 796, 10178, 10199, 10301, 10607, 10685, 10707,
  11503, 11600, 11665, 11690
- SCC
  - ICPC 4262, 4272
  - POJ 2186
  - Uva 111504 , 11709, 11710, 11838