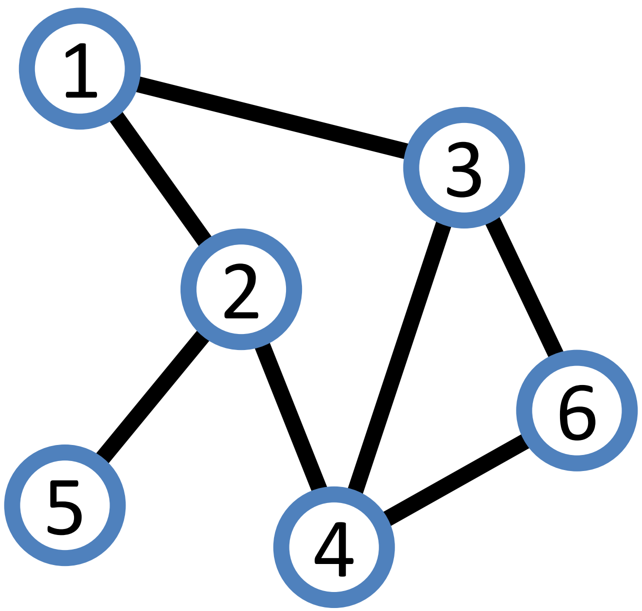
Elementary Graph Algorithms
Chapter 22

Mei-Chen Yeh

Graph (1)



$V = \{1, 2, 3, 4, 5, 6\}$ → node

$E = \{(1,2)\ (1,3)\ (2,4)$
$(2,5)\ (3,4)\ (3,6)\ (4,6)\}$ 每條邊是連接哪兩個節點 E = {(邊1) (邊2)}

- $G = (V, E)$

vertex　　edge

or
node

|V|：裡面有幾個元素

- $|V|$: number of vertices
- $|E|$: number of edges
- ***Sparse*** graphs

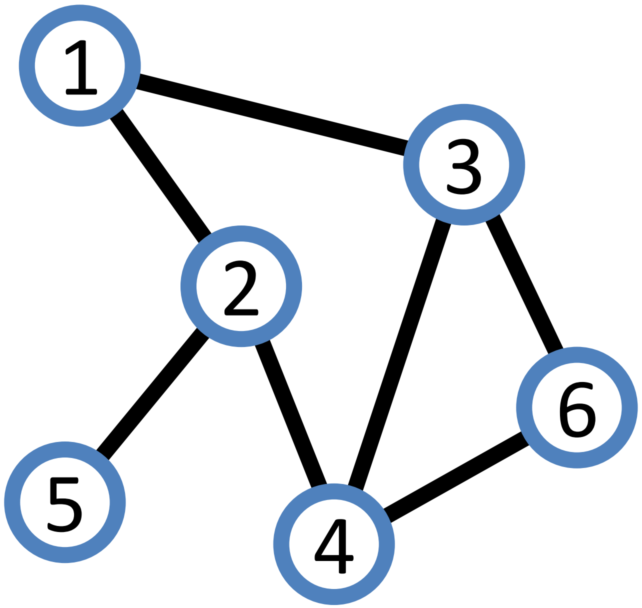
  $|E| << |V|^2$

  邊的個數vs節點數的平方 來判斷是不是sparse
- ***Dense*** graphs

  $|E| \approx |V|^2$

Some slides are modified from Prof. Lu's slides.

Graph (2)



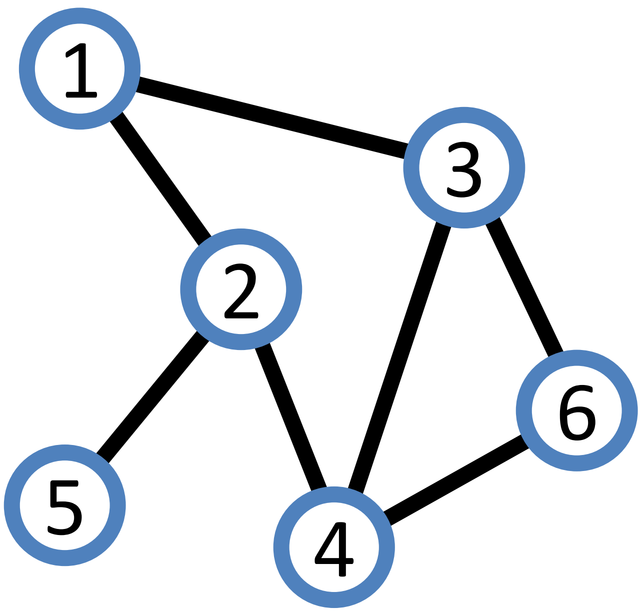
Degree：這個節點有幾個連居（有幾個邊碰到它）

- ***Degree*** of a node
  - \# of neighbors
- ***Degree*** of the graph

  max（每個節點的degree）
  - maximum degree over all of its nodes
- ***Directed*** or ***undirected***?

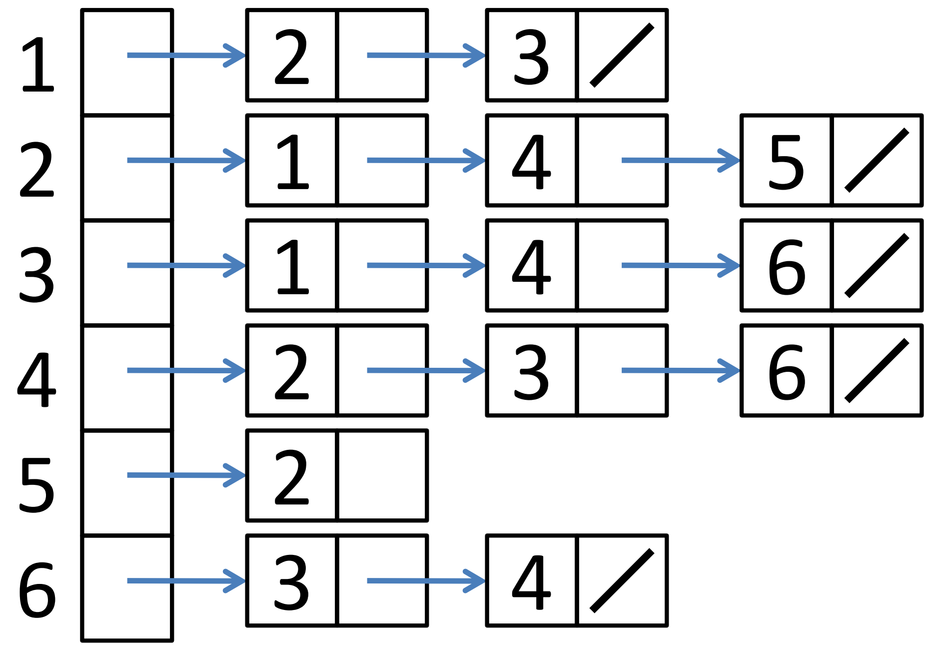
Directed graph：看有沒有箭號指向某個節點

Representations of graphs (1)



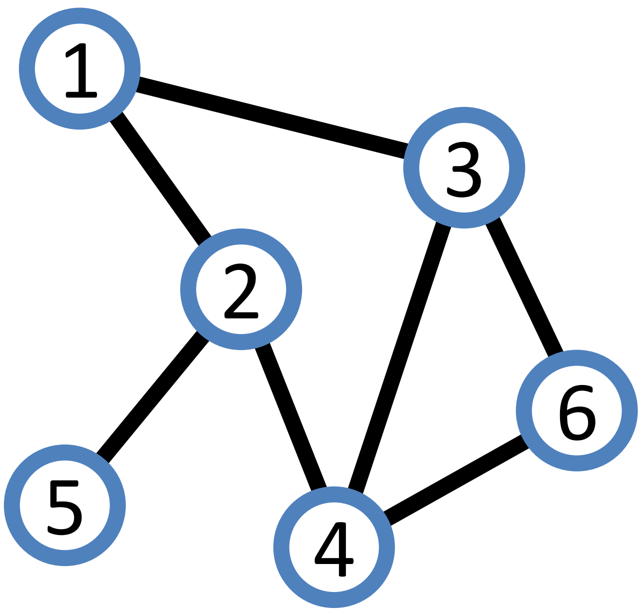
linked list

Adjacency-list



建立一個一維的array，節點指向下一個要去的節點

Representations of graphs (2)



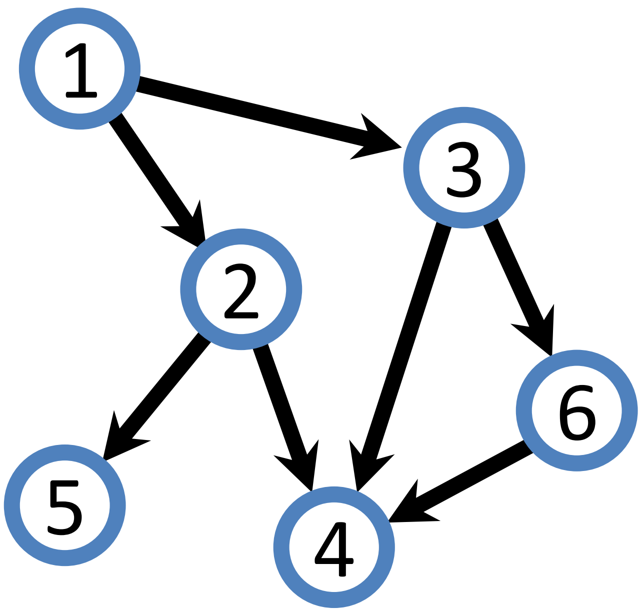
宣告一個二維的array（6個節點故宣告6x6的大小）

Adjacency-matrix

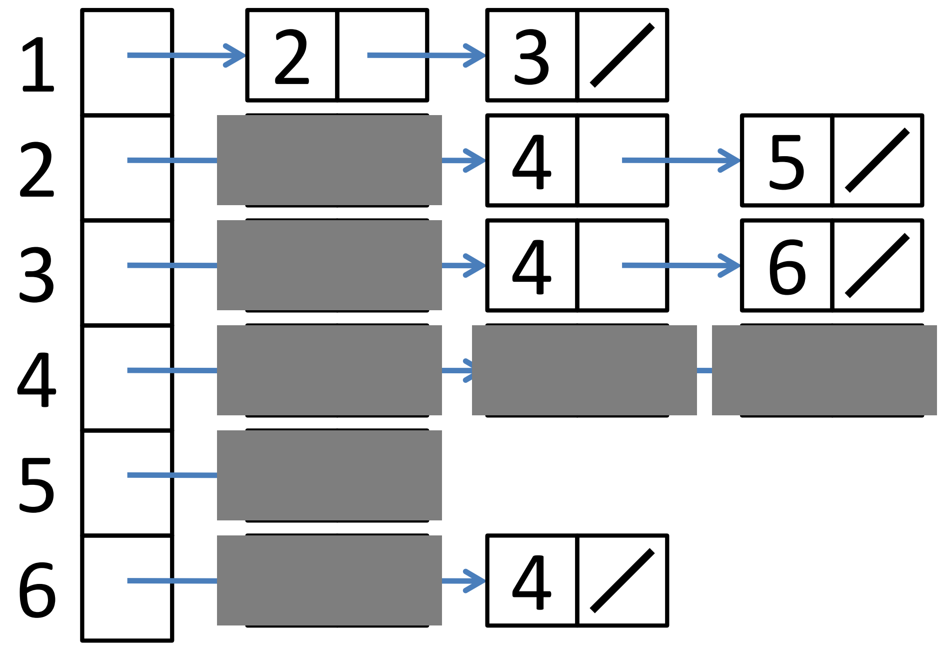
	1	**2**	**3**	**4**	**5**	**6**
1		1	1			
2	1			1	1	
3	1			1		1
4		1	1			1
5		1				
6			1	1		

如果是比較sparse的情況，就不要用matrix，用
linked list比較合適（才不會浪費空間存一堆null）

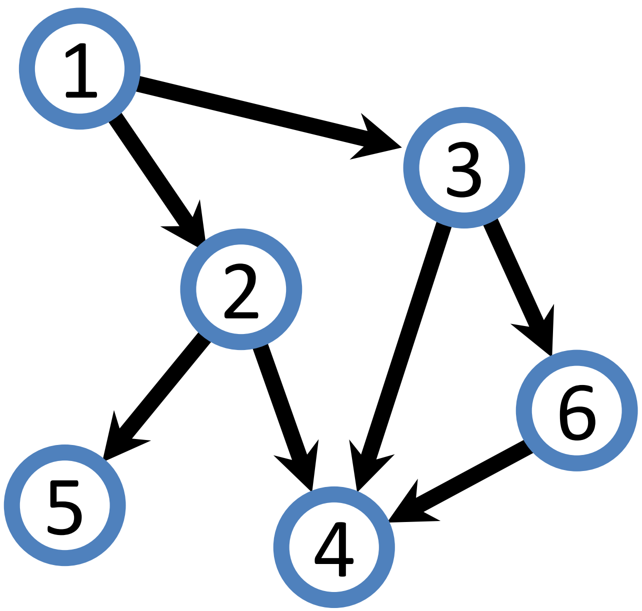
Representations of graphs (3)



Adjacency-list



Representations of graphs (4)



Adjacency-matrix

	1	2	3	4	5	6
1		1	1			
2				1	1	
3				1		1
4						
5						
6				1		

不會是對稱的

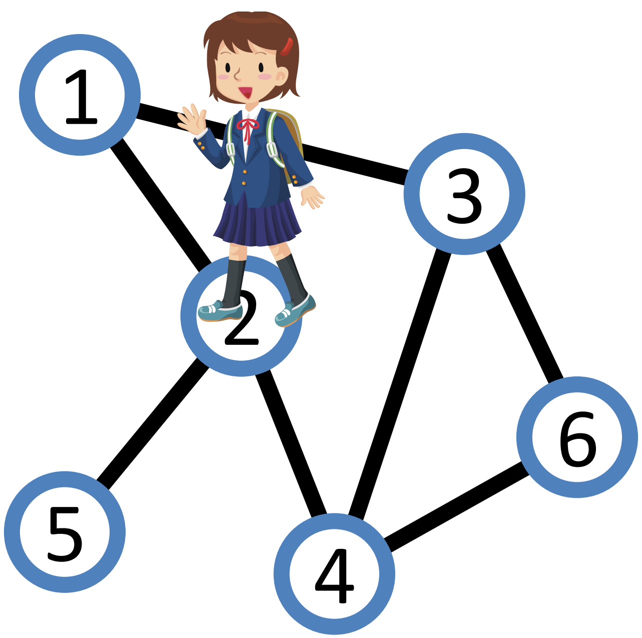
Graph traversal

想讓小人有系統性地去拜訪每一個節點

- Breadth-first search (BFS) 廣度優先
- Depth-first search (DFS) 深度優先

較常用

A systematic way to visit vertices and to traverse edges of a graph.



Breadth-first search

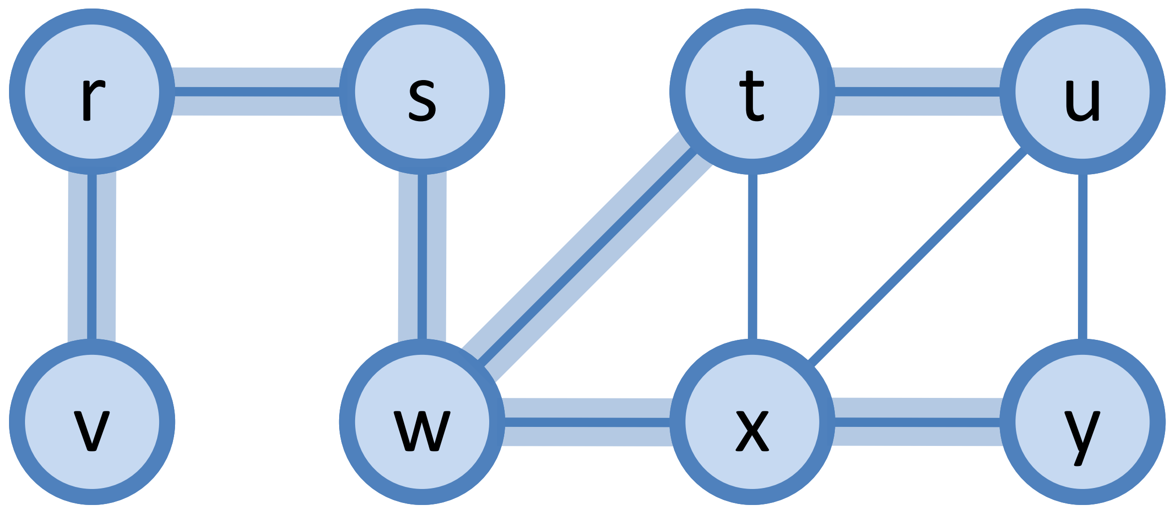
Breadth-first search

BF tree

Breadth-first tree

x is y's ***parent.***

s是r,w的parent



把小人放進s節點

把已經放進來的節點的每個鄰居都放進來

Queue

先進先出

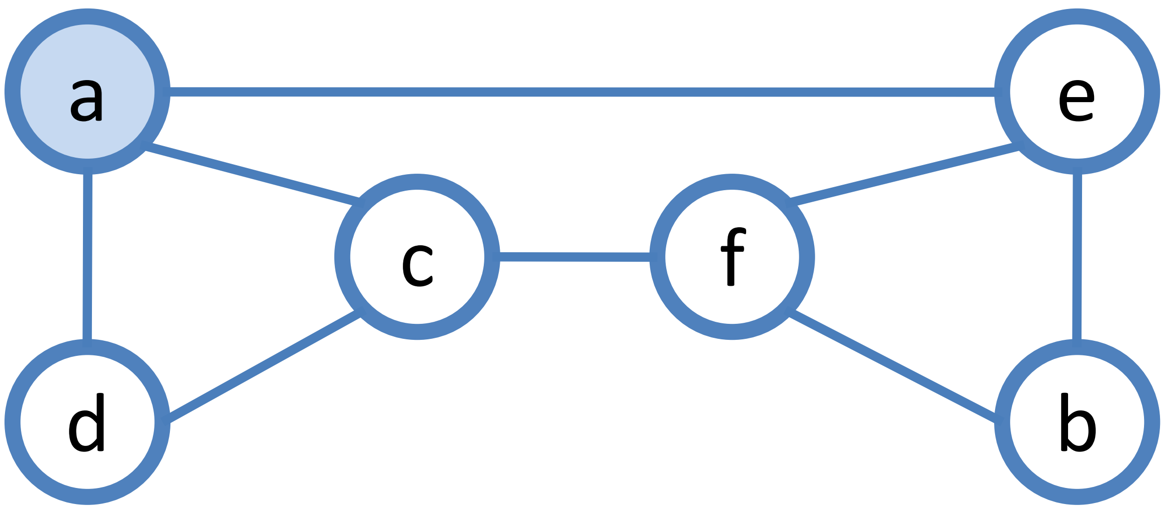
s	w	r	t	x	v	u	y

***discovery** time*

講到BFS就先建立queue，因為需要queue來實現BFS

Exercise

What is the discovery order of vertices?
Resolve ties by the vertex alphabetical order.



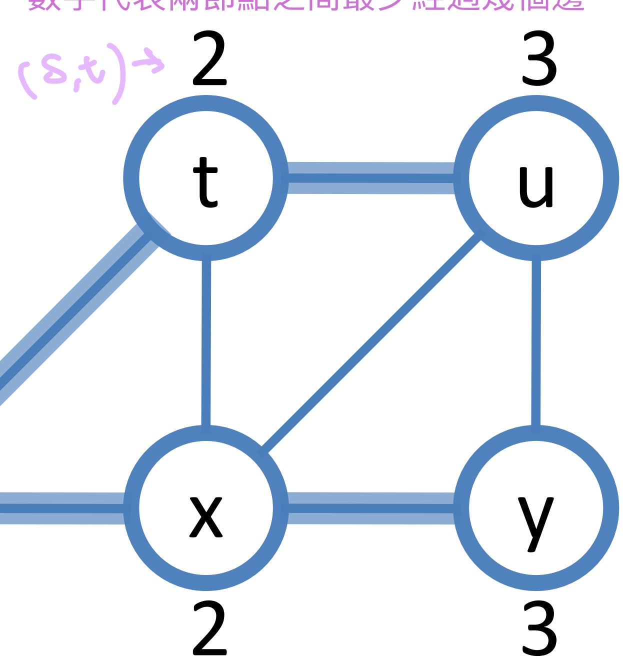
acdefb

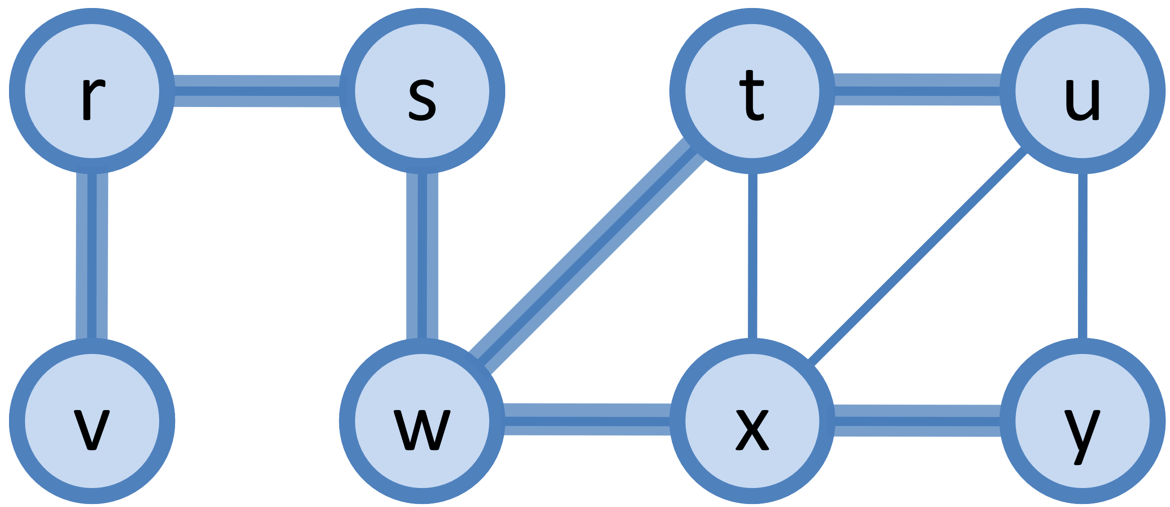
Shortest paths

最短路徑問題

- $\delta(s, v)$: the minimal number of edges in any path from vertex s to vertex v

數字代表兩節點之間最少經過幾個邊





Queue	s	w	r	t	x	v	u	y
v.d:	0	1	1	2	2	2	3	3

把每個節點的最短路徑蕾加起來

Depth-first search

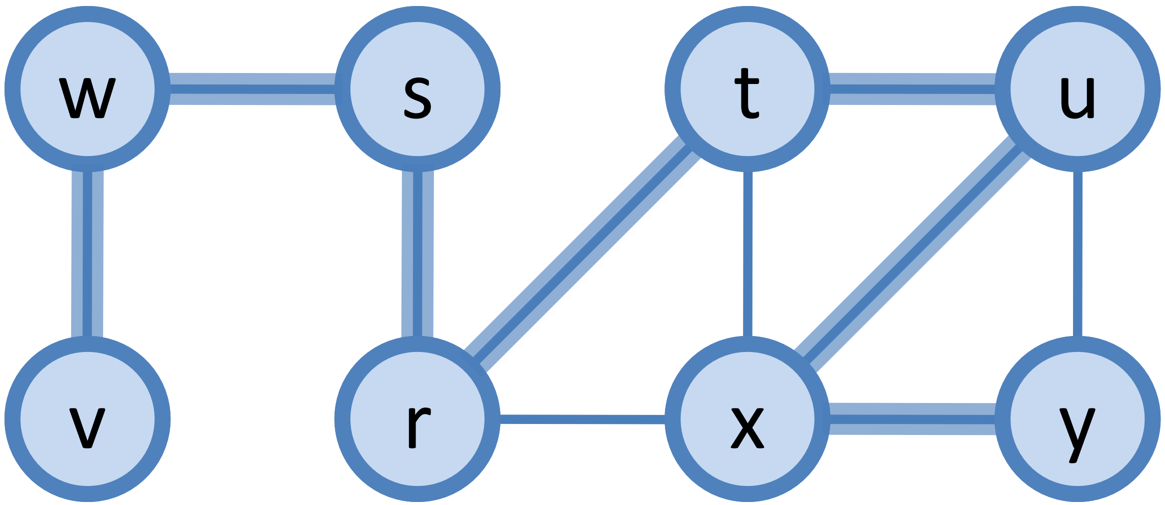
Depth-first search

深度優先：找一條路，走到底，有多遠走多遠

DF tree

depth-first tree

走到不能再走：現在這個節點的所有鄰居都拜訪過了，表示不能再走了



***finishing** time*

有兩個不同的時間，
路徑不完全相反

Stack

s	r	t	u	x	y	w	v

先進後出

結果可能有好幾種

***discovery** time*

Depth-first search + time stamps

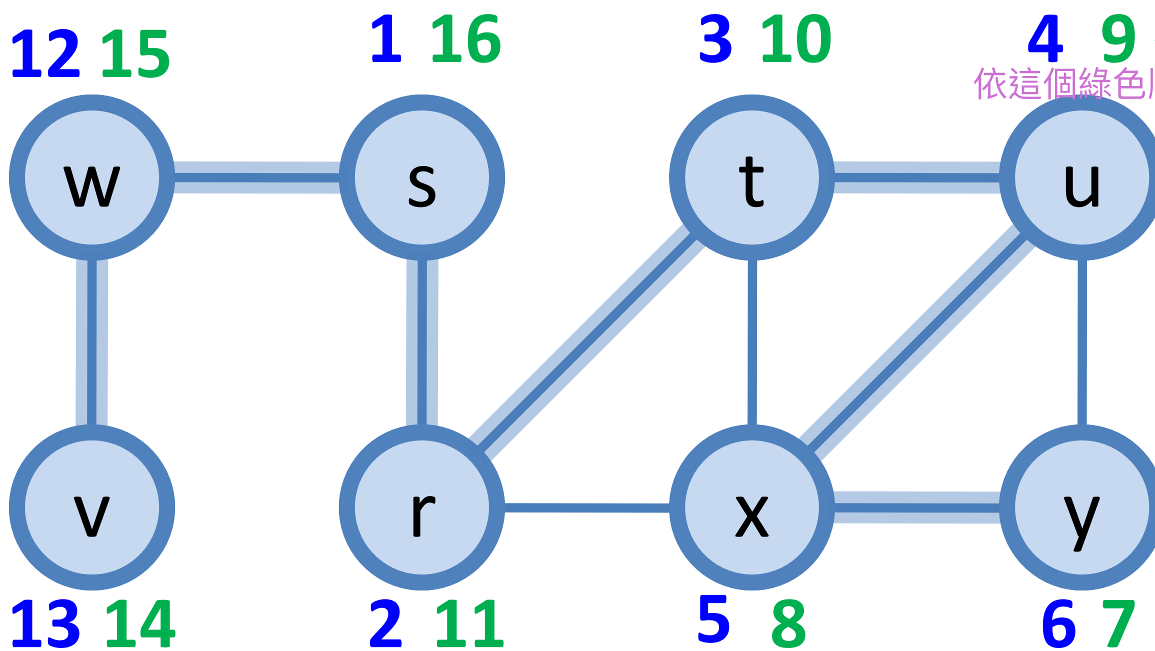
按照順序放進stack

discovery time

依這個綠色順序按照時間出stack

finishing time

路線不是唯一的可能



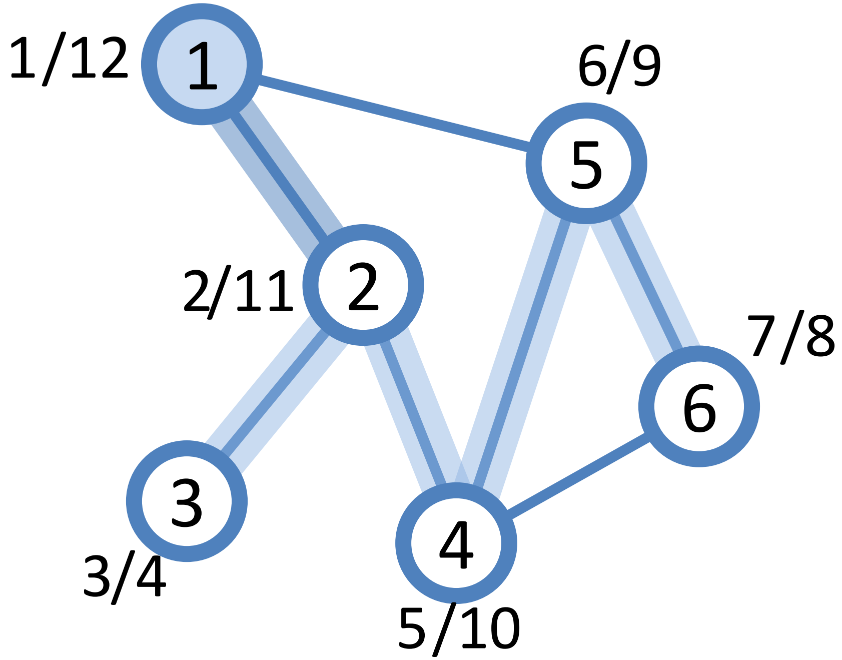
Stack

s	r	t	u	x	y	w	v

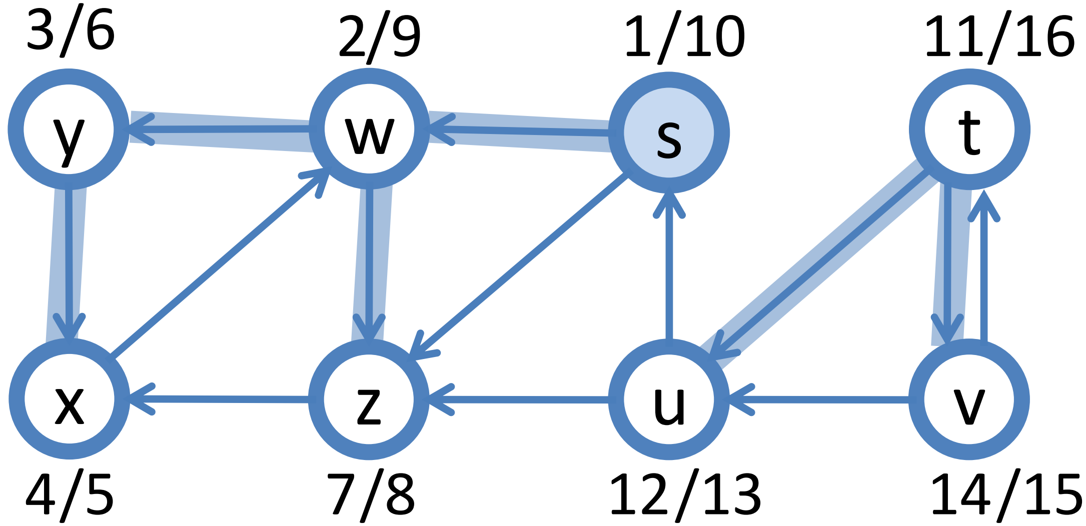
先進後出

Exercise

Find the discovery/finishing time of each node.



Exercise



Properties (1)

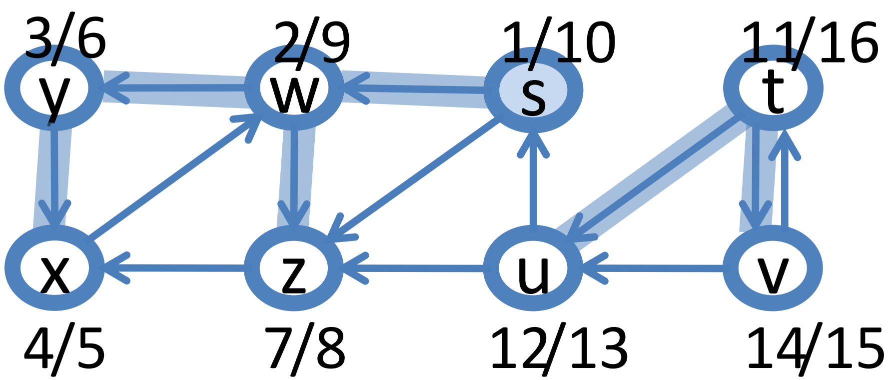
- Discovery and finishing times have ***parenthesis structure***.

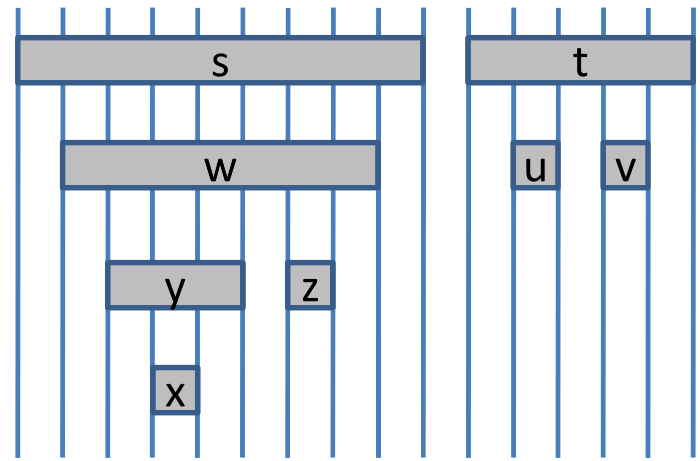
不會有overlap的情況，要嘛沒交集，
要嘛完全不重疊（大包小）

重疊的情況只有兩種可能性

Entirely disjoint!

Entirely within!





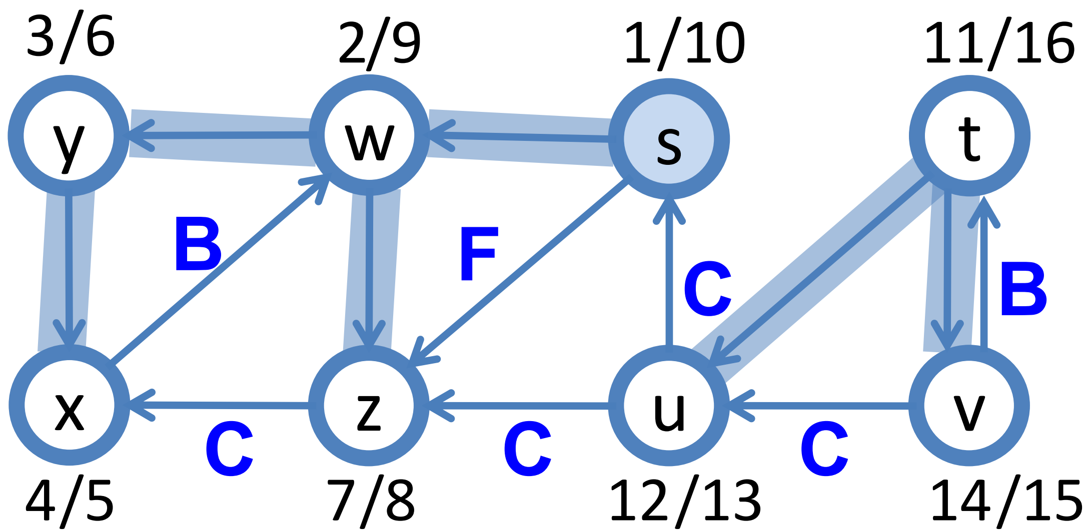
(s (w (y (x x) y)(z z) w)s)(t (u u) (v v) t)

Properties (2)

把edge做分類

- Classification of edges: tree (**T**), back (B), forward (F), and cross (C)

Tree：一代傳一代
Back：箭頭指回親代
forword：隔代，跳過了自己的上一代
cross：
有同樣的祖先，
但彼此間沒有直接箭頭



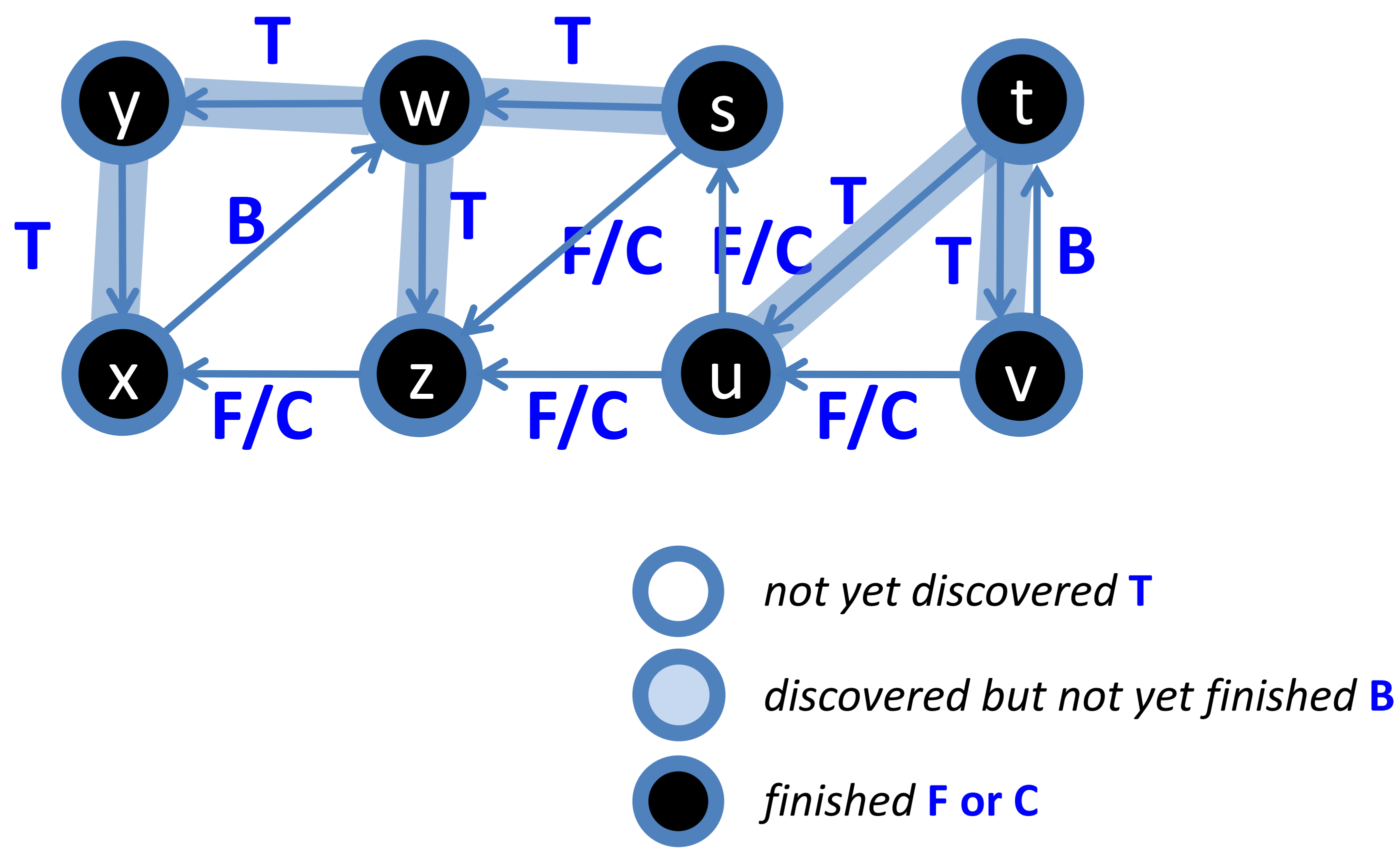
Properties (2) $u \longrightarrow v$

- The DFS algorithm classifies some edges as it encounters them.
- How? Look at the state of *v*:
  - *not yet discovered* → tree edge
  - *discovered but not yet finished* → back edge
  - *finished* → forward or cross edge

Properties (2)

- Example:

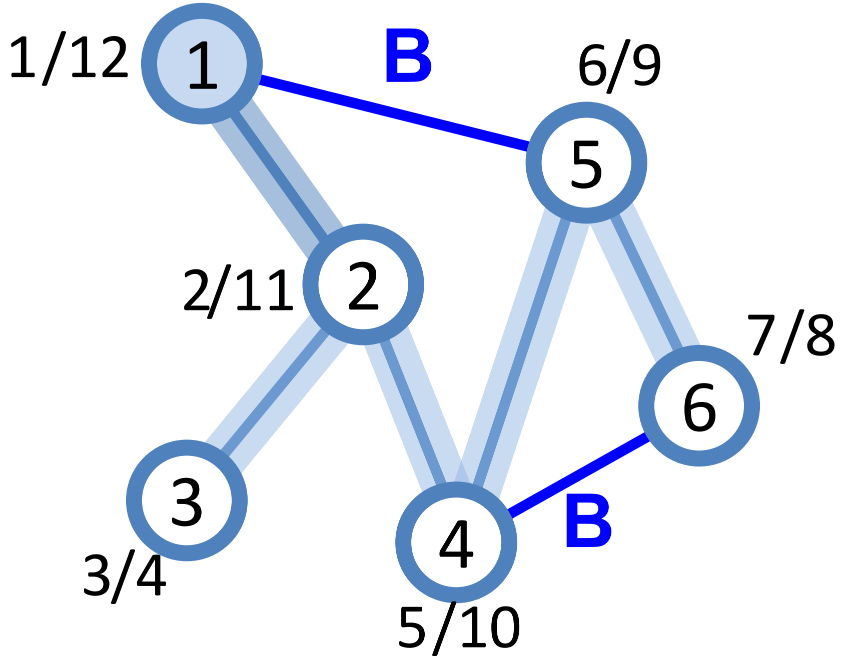
使用DFS在走的時候可以順便把節點做分類



Properties (2)

- In a depth-first search of an ***undirected*** graph, every edge is either a ***tree*** edge or a ***back*** edge.

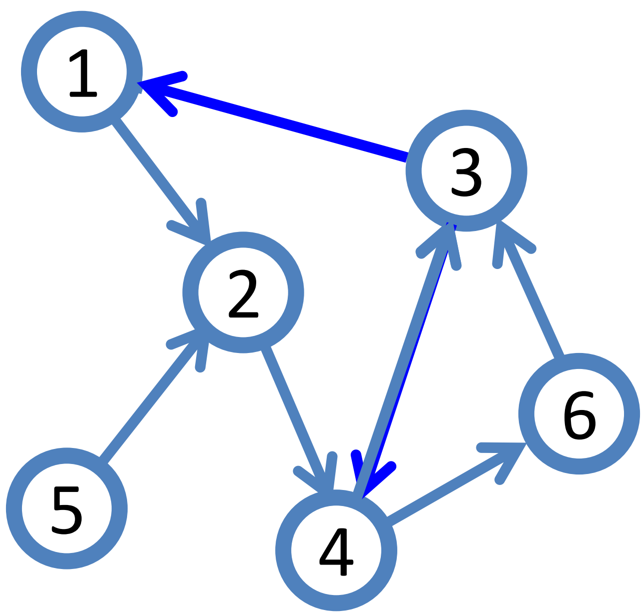
如果今天edge之間沒有箭頭指向，那節點只會剩下兩個種類 tree 跟 back



Directed acyclic graph (DAG)

- DAG: A directed graph that ***does not contain any directed cycle***.

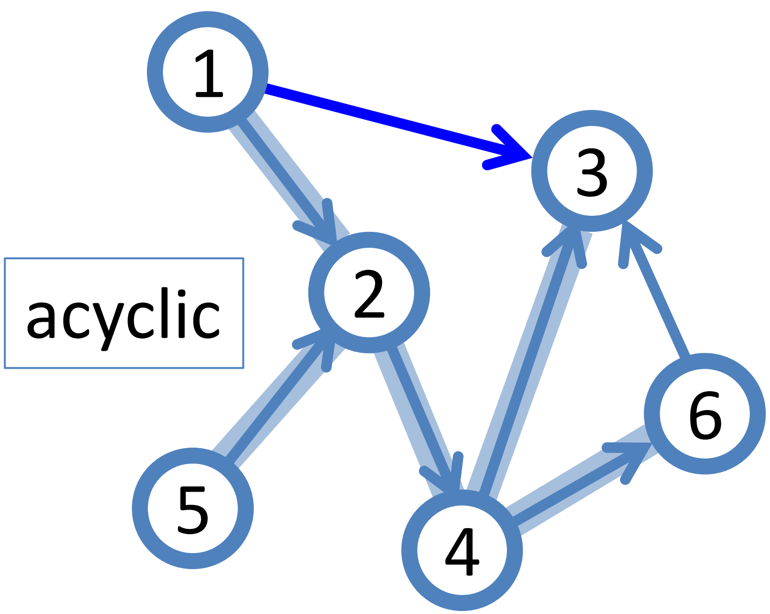
DAG：在這個graph裡面沒有任何的圓環

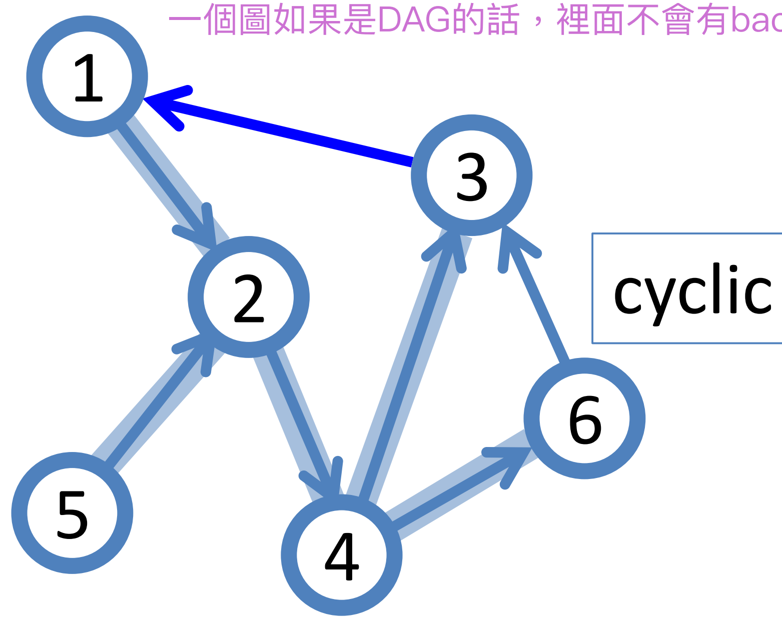


Exercise

- How to know if a graph *G* is (or is not) a dag?
- Hint: A directed graph is acyclic if and only if a DFS has no <u>back</u> edges!

一個圖如果是DAG的話，裡面不會有back edge



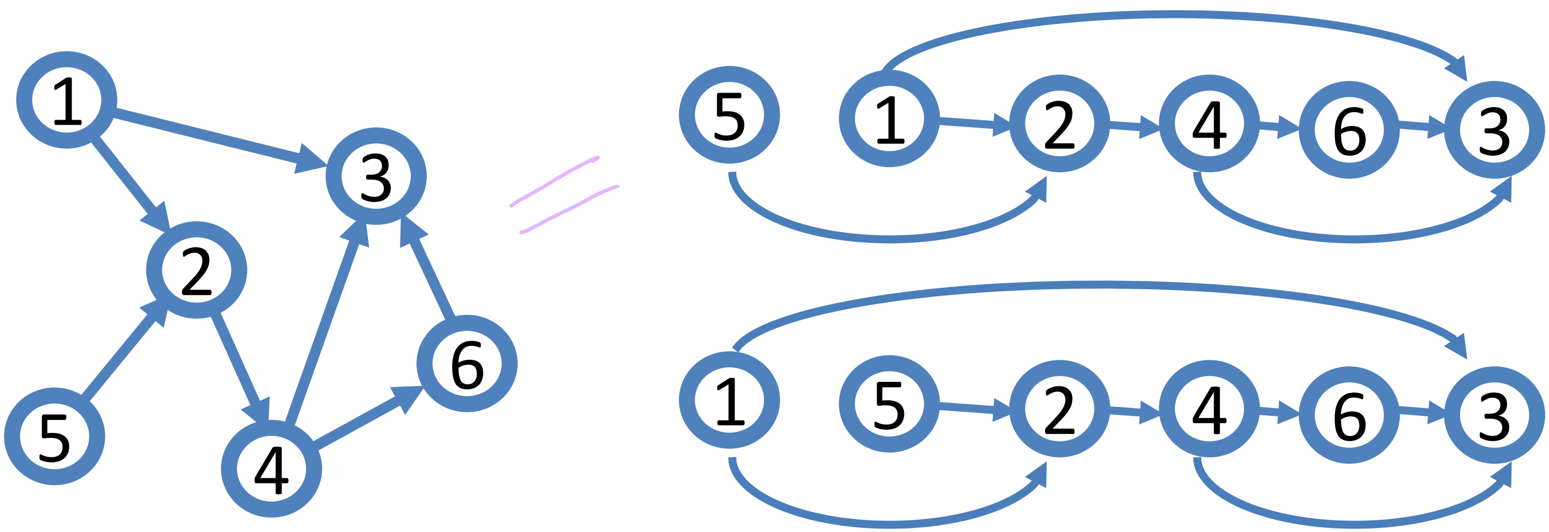


Topological sort

只有是DAG的input才能做topological sort

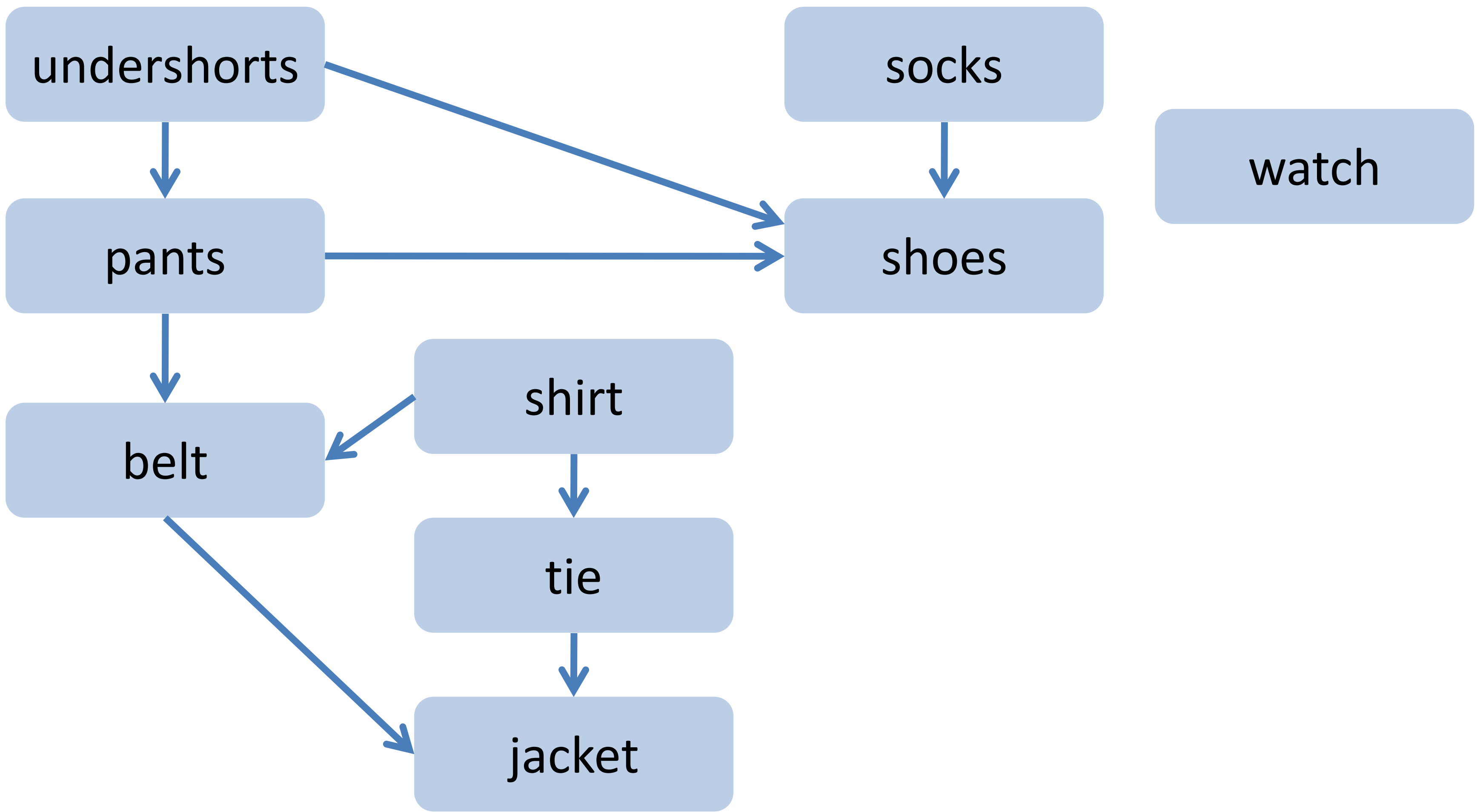
- Given a **dag** $G = (V, E)$, find a linear ordering of all its vertices such that if G has an edge (u, v), then u appears ***before*** v.

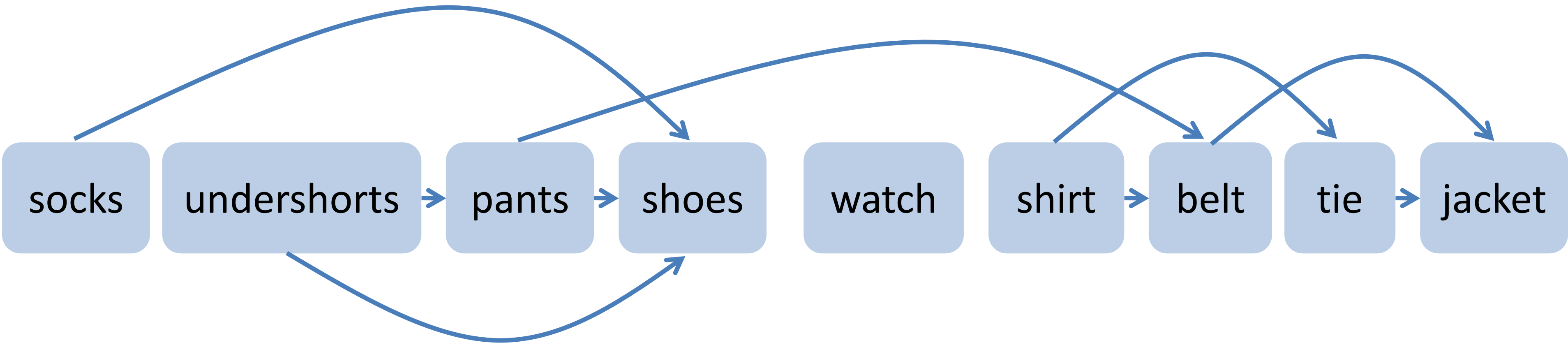
調整節點的順序
讓排前面的往後指



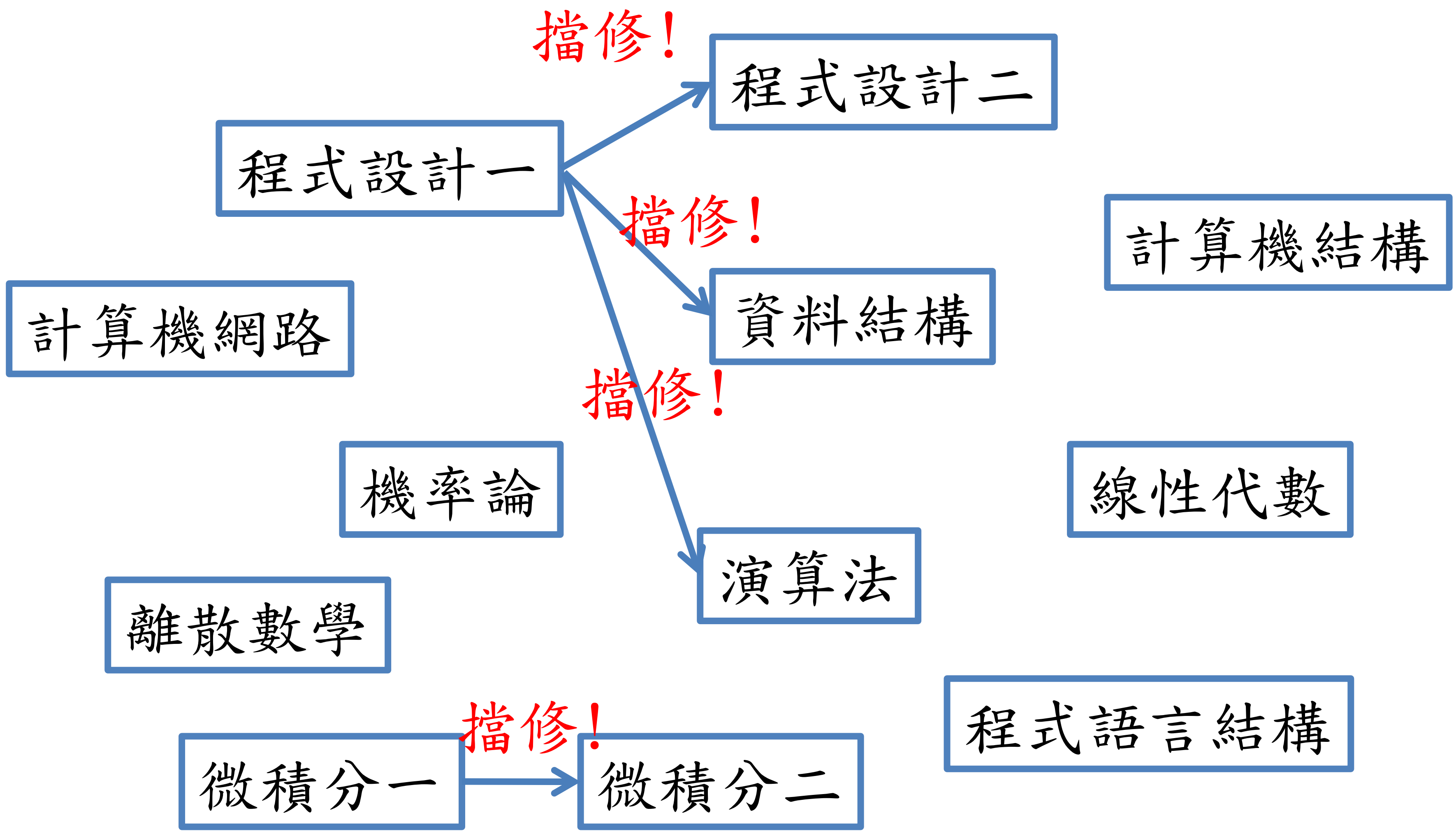
Example (1)

有個老先生，每天起床都要穿戴下面的物品才願意出門



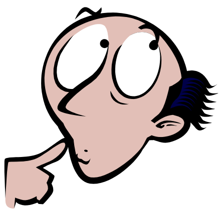

socks
undershorts
pants
shoes
watch
shirt
belt
tie
jacket

Example (2)



擋修！
程式設計二
程式設計一
擋修！
計算機結構
計算機網路
資料結構
擋修！
機率論
線性代數
演算法
離散數學
擋修！
程式語言結構
微積分一
微積分二

How to solve the topological sort problem?



- Use the finishing time by DFS!

按照finishing time來排序

- 2-step procedure:

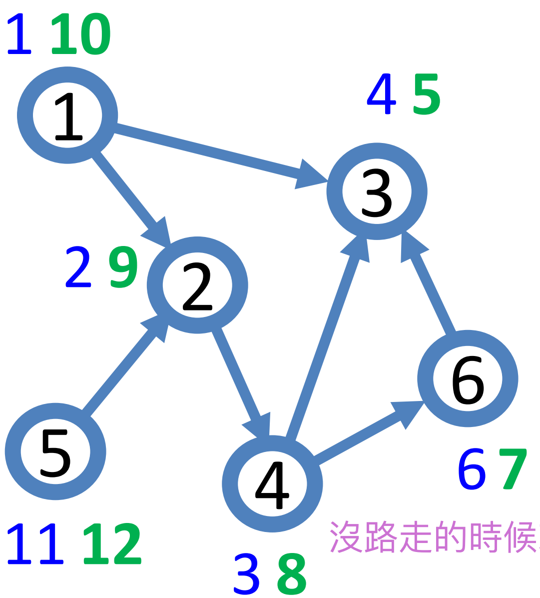
按照節點出來的時間，從大的排到小的

  1. Run DFS
  2. Output the nodes in decreasing order of their finishing time.

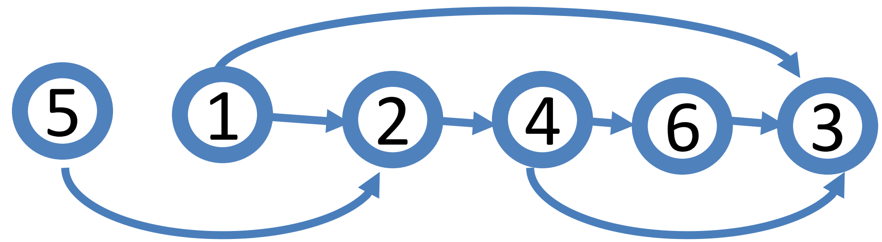
不是唯一解，選一條路走到底即可

Example

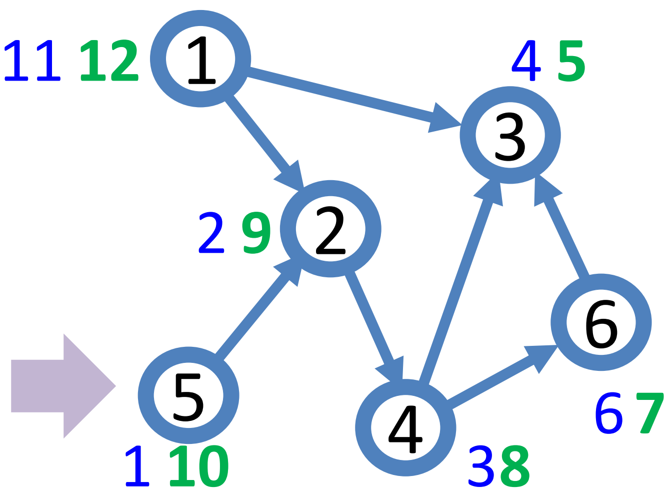
走法1

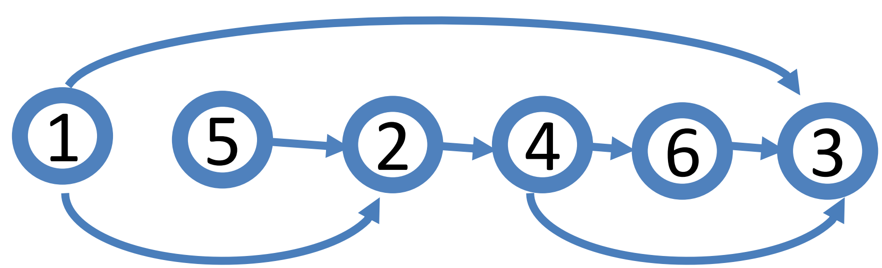


沒路走的時候就在自己身上+1，然後回頭



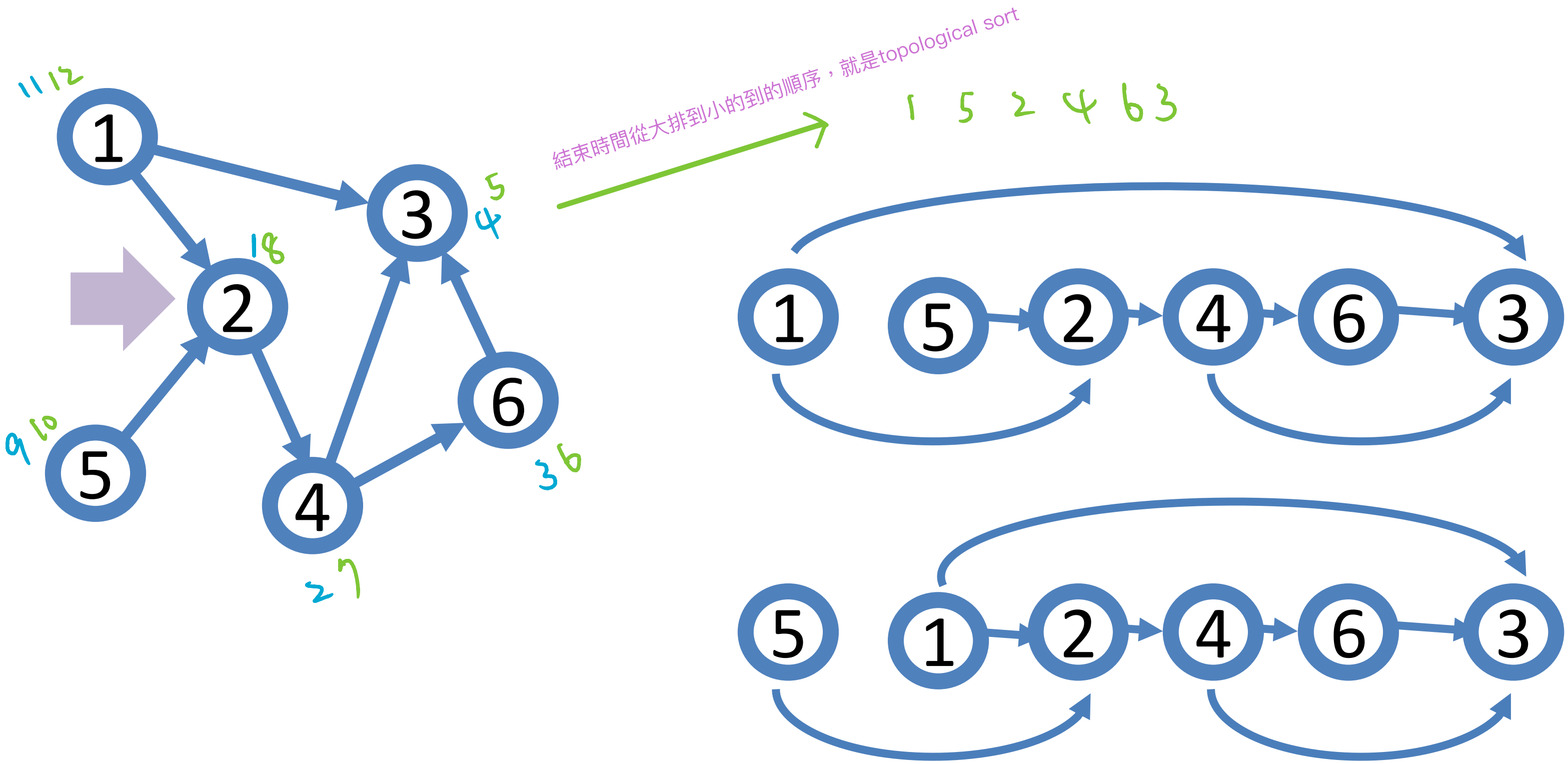
走法2





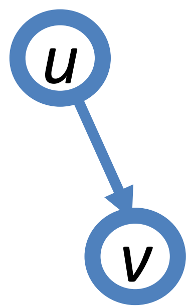
Exercise

- What's the ordering if we start from vertex 2?



Correctness (1)

- Show that if (u, v) is a directed edge of G, then

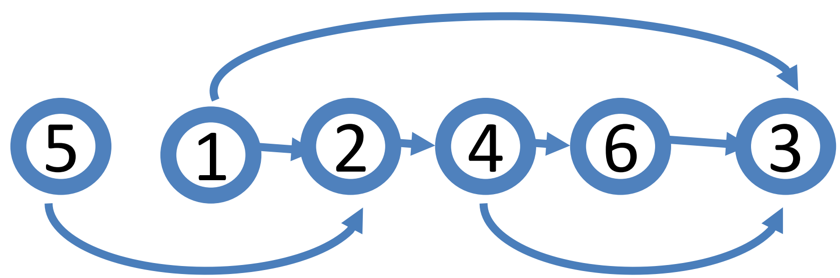


$u.f$ __>__ $v.f$

hint: > = <

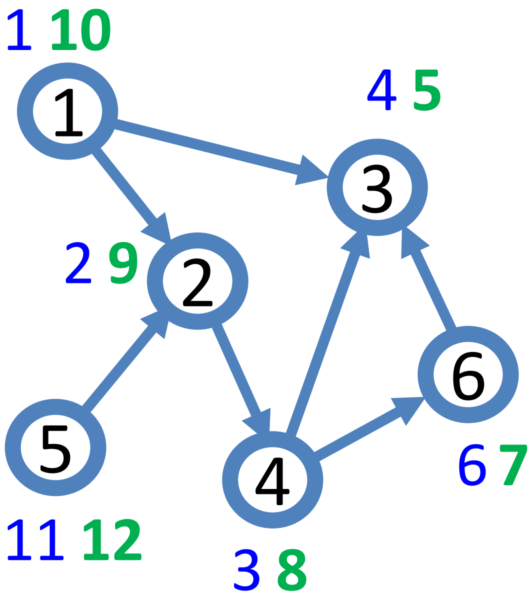
Q

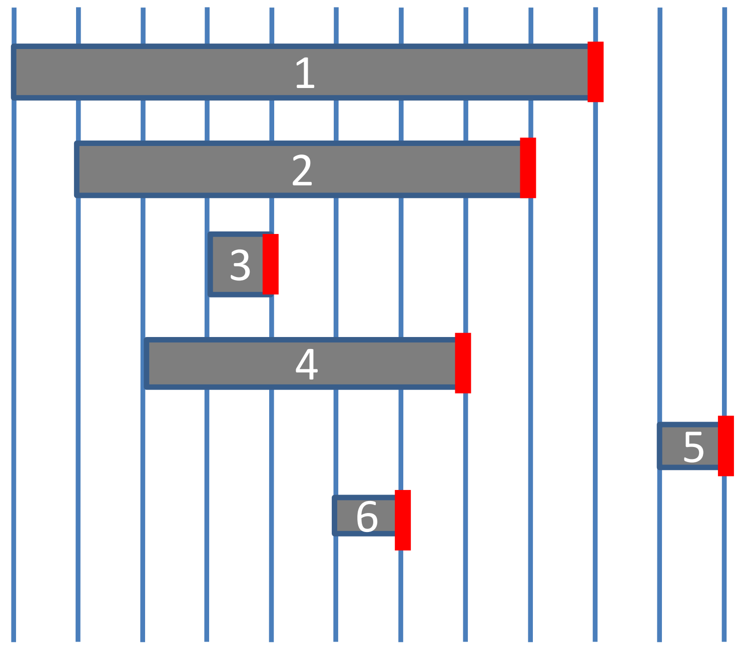
為什麼按照finishing time從大排到小的順序，就是topological sort的順序？



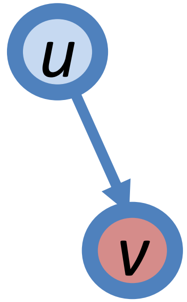
finishing time (f) ⟵ ·························

large small



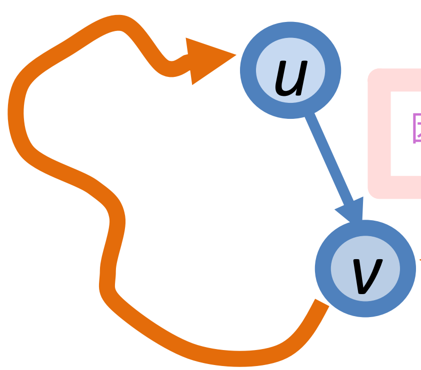


$$\text{if } u \longrightarrow v, \quad u.f > v.f$$



Correctness (2)

- Show that if (u, v) is a directed edge of G, then $u.f > v.f$.
  - *– v is not yet discovered, u.f > v.f (why?)*
  - *– v is finished, u.f > v.f (why?)*
  - *– v is discovered but not yet finished?*
    - **Impossible! Otherwise this is not a DAG!**



因為這種情況會變成圓環，他就不是DAG，就不能做topological sort

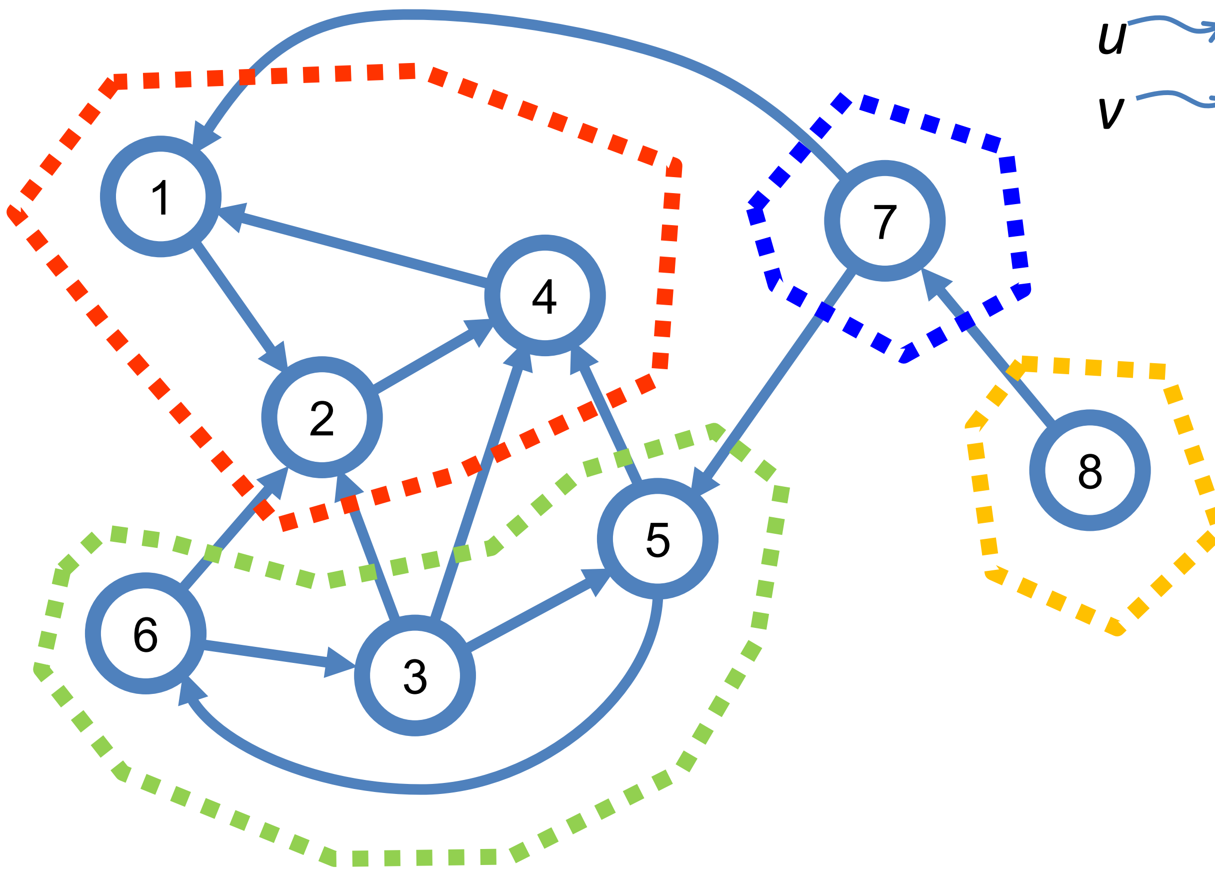
Strongly connected components

- Decompose a directed graph into its ***strongly connected components***.

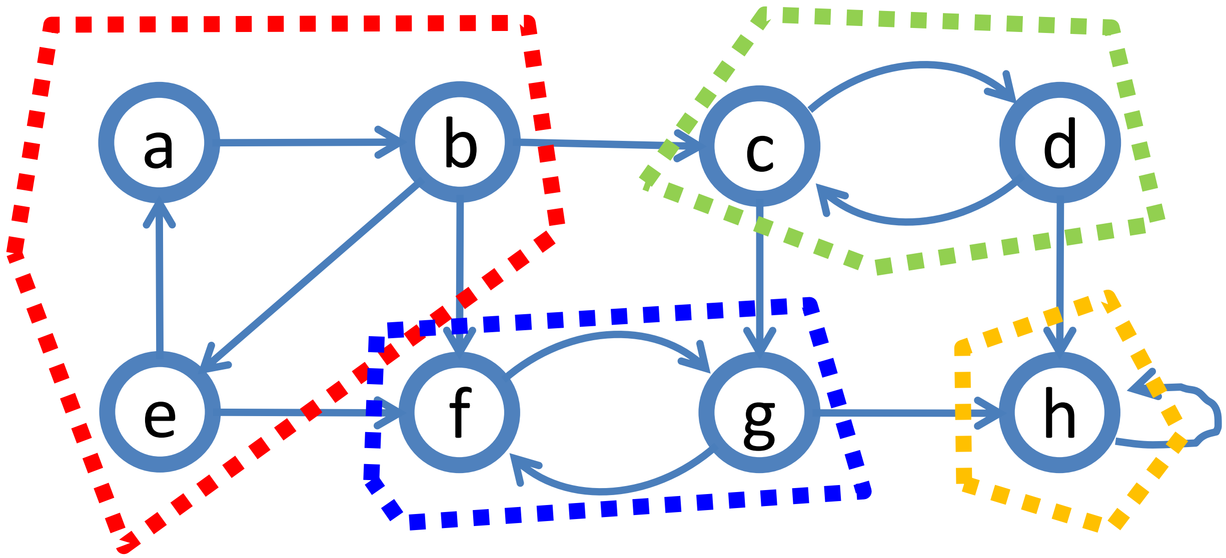
for every (u, v) in C_i

$u \rightsquigarrow v$

$v \rightsquigarrow u$

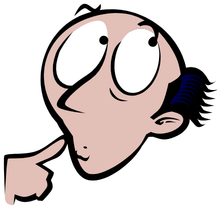


One more example



Strongly connected components

- Input
  - $G = (V, E)$
- Output
  - The strongly connected components of G

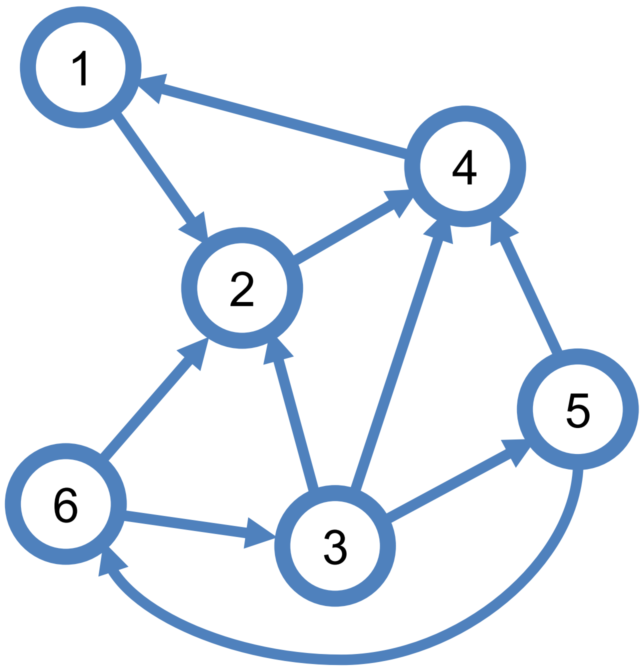


How to solve the problem?

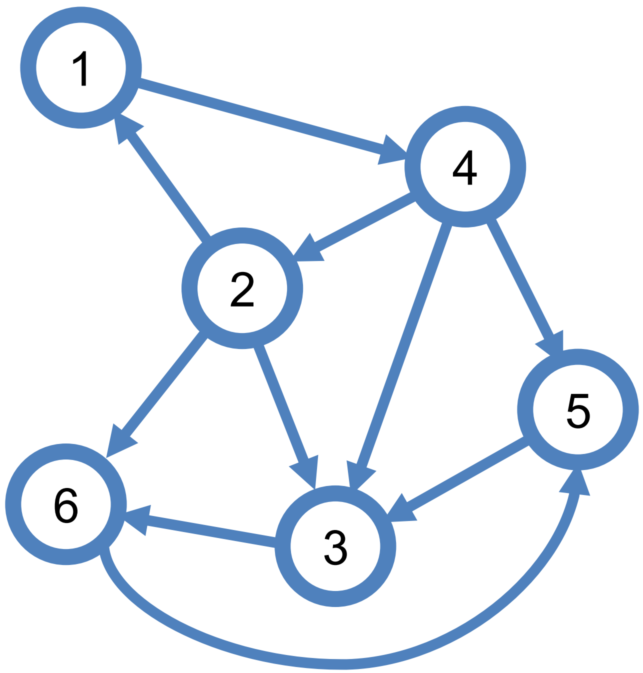
A linear $\Theta(V+E)$-time algorithm

1. call DFS(G) to compute finishing time $u.f$ for each vertex u
2. compute G^{T} (the transpose of G)
3. call DFS(G^{T}) with considering the vertices in the order of decreasing $u.f$
4. output the components found in step 3

Graph transpose

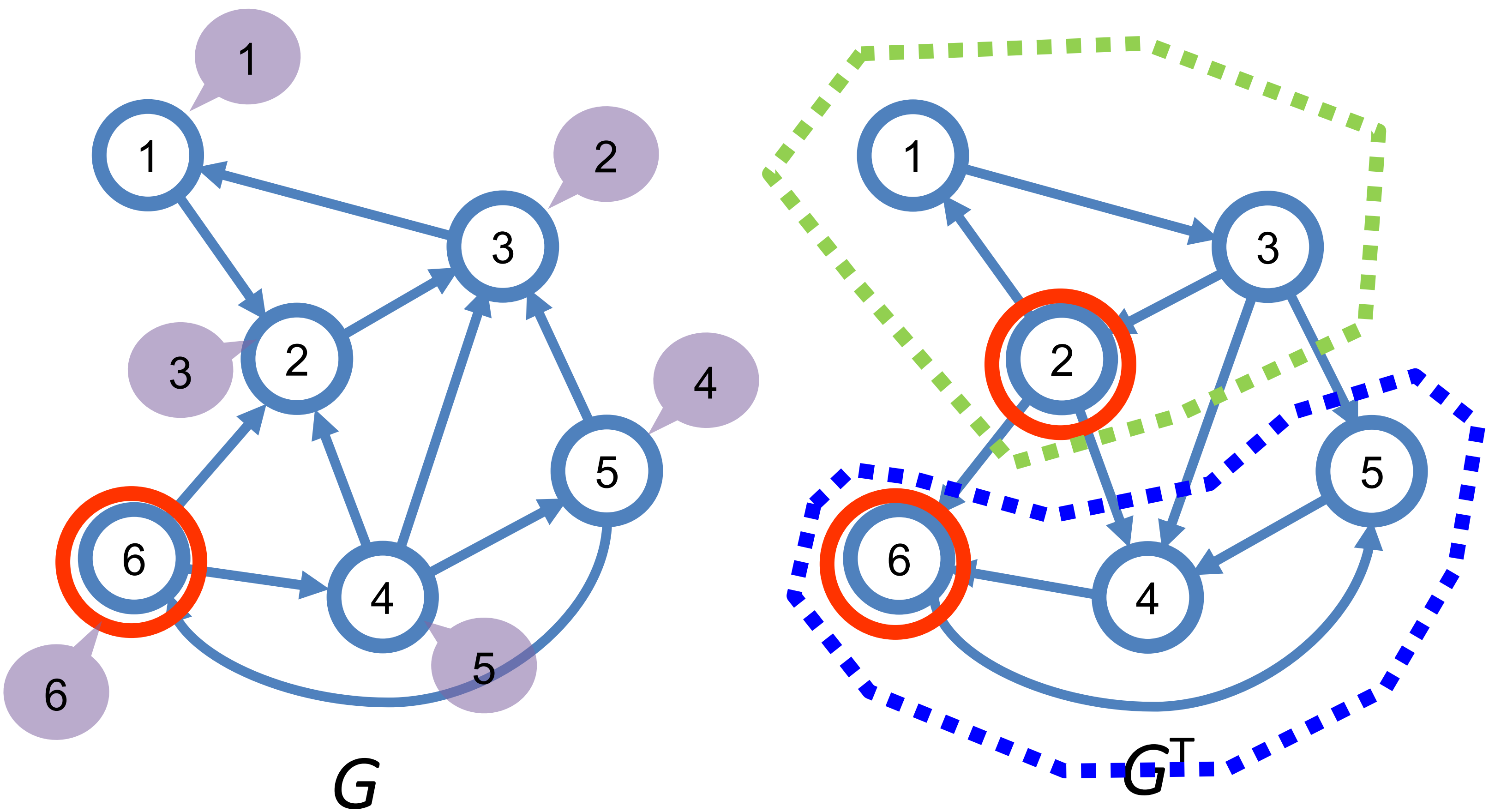


G

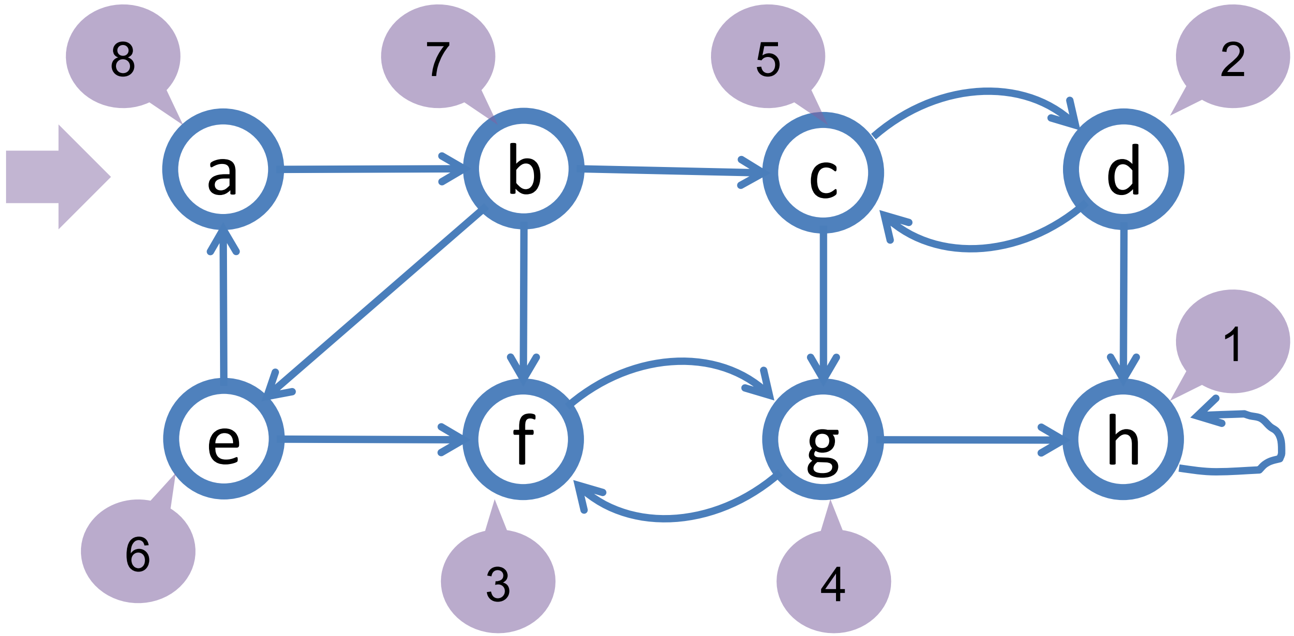


G^{T}

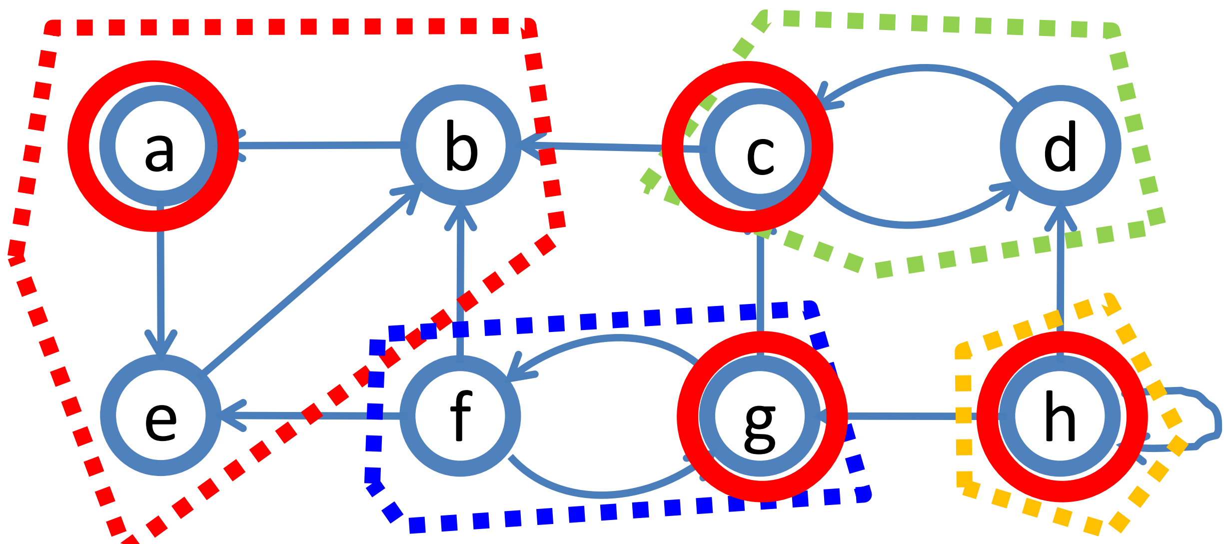
Example



做GT，拷貝一份，所有的邊做反向。
從結束時間最大開始



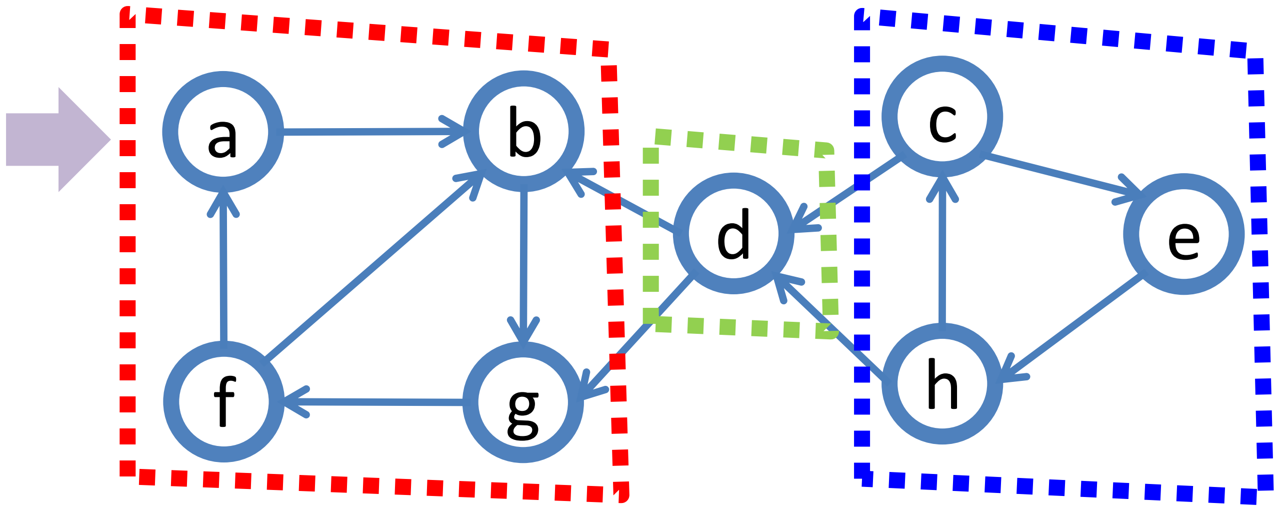
從結束時間最大的節點開始做DFS，發現跑不出去，找到strongly connected component



可以找到 4 組

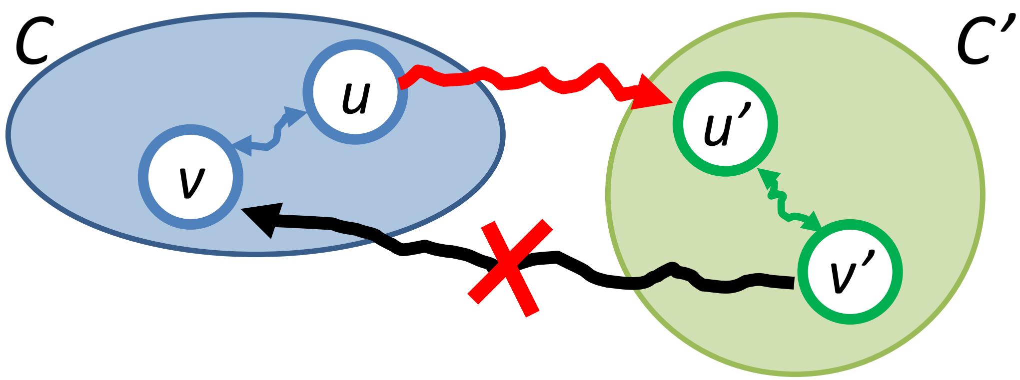
Exercise

- Apply DFS to find its strongly connected components.



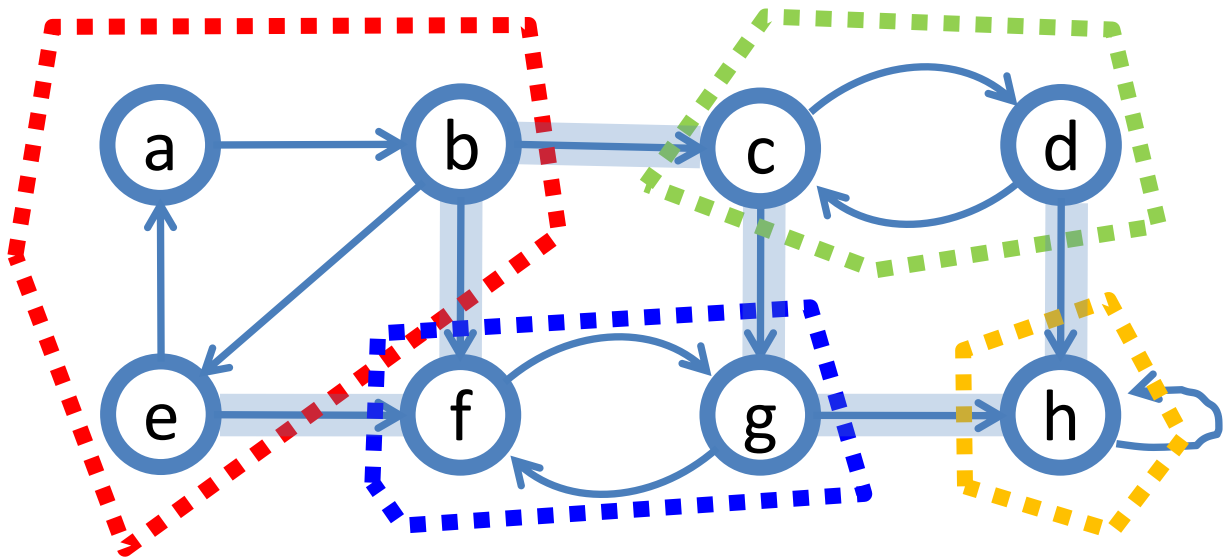
Correctness

- Observation 1



If G contains a path $v' \to v$, then it contains paths $u \to u' \to v'$ and $v' \to v \to u$. Thus u and v' are reachable from each other! That contradicts the assumption that C and C' are distinct strongly connected component.

Example

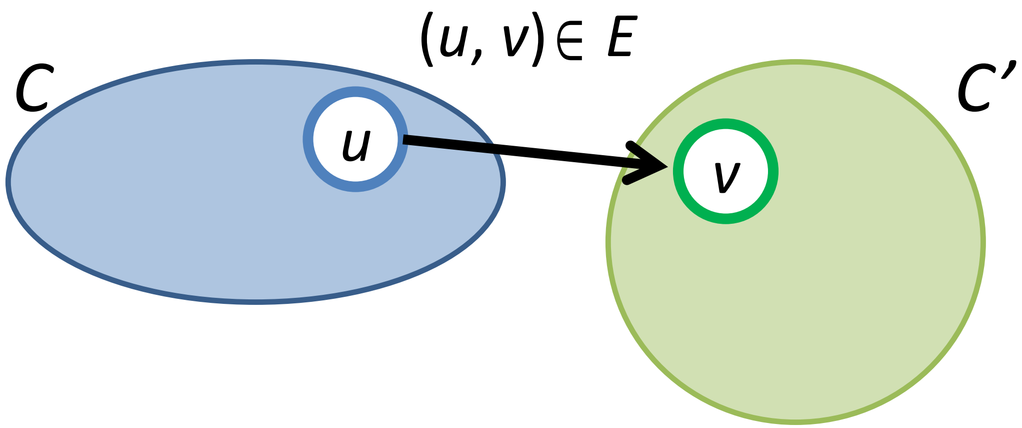


Correctness

$f(C) = \max\{u.f\}$
最後結束的時間

$d(C) = \min\{u.d\}$
最早開始的時間

- Observation 2



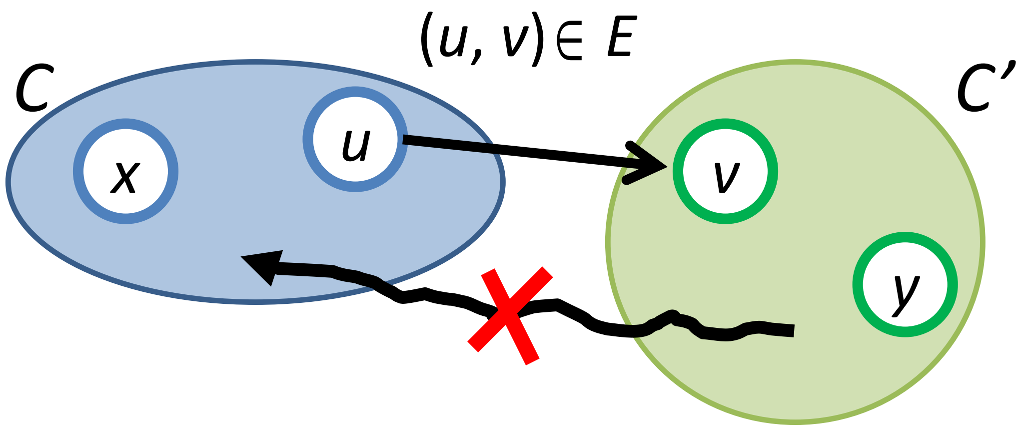
$f(C)$ **>** $f(C')$ ***(why?)***

Correctness

$f(C) = \max\{u.f\}$
最後結束的時間

$d(C) = \min\{u.d\}$
最早開始的時間

- Observation 2



$f(C)$ **>** $f(C')$ **(why?)**

2 cases:

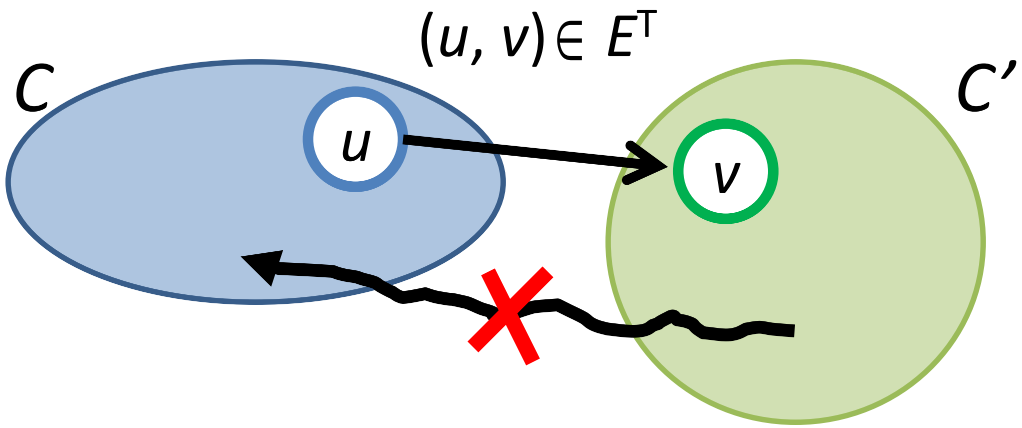
- if $d(C) < d(C')$
- if $d(C) > d(C')$

Correctness

$f(C) = \max\{u.f\}$
最後結束的時間

$d(C) = \min\{u.d\}$
最早開始的時間

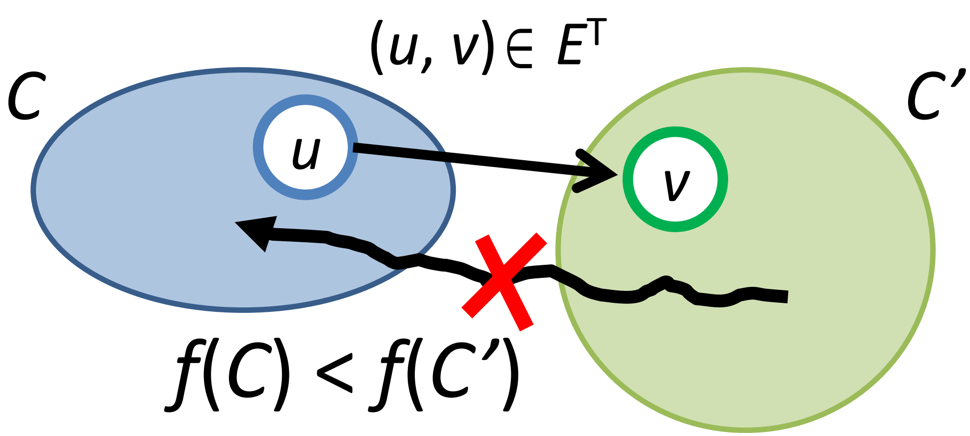
- Observation 3



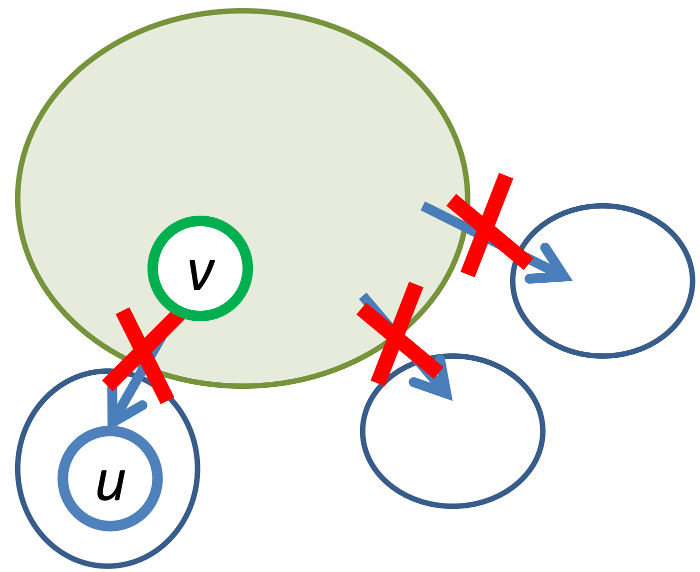
$f(C) < f(C')$

Correctness

1. call DFS(G) to compute finishing time $u.f$ for each vertex u
2. compute G^T (the transpose of G)
3. call DFS(G^T) with considering the vertices in the order of ***decreasing*** $u.f$
4. output the components found in step 3

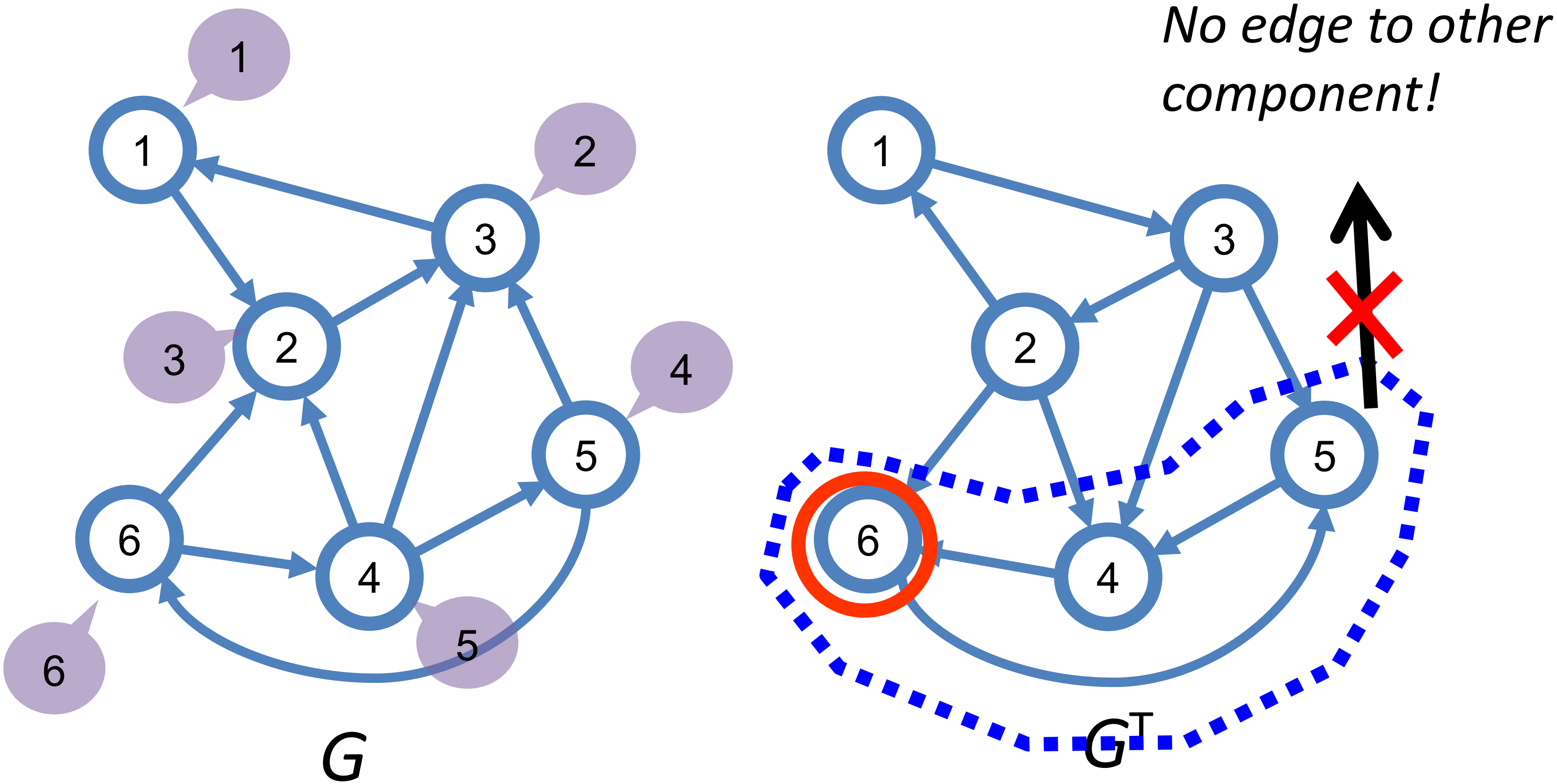


C': has the **max** $\boldsymbol{f(C)}$



E^T contains ***no edges*** from C' to any other strongly connected component!

Example



Summary

- Breadth-first search (BFS)
  - shortest path between two vertices
- Depth-first search (DFS)
  - topological sort
  - strongly connected components