連載: 學生上課睡覺姿勢大全

http://www.wretch.cc/blog/chi771027/26489957

GRAPH 2

Michael Tsai

2013/4/23

Breadth-First Tree

- BFS做出一棵樹: Breadth-First Tree
- 這個樹其實也就是BFS產生出來的predecessor subgraph of G:
- $G_\pi = (V_\pi, E_\pi)$
- $V_\pi = \{v \in V : v.\pi \neq NIL\} \cup \{s\}$
- $E_\pi = \{(v.\pi, v) : v \in V_\pi - \{s\}\}$
- 從s到任何$v \in V_\pi$, 都只有一條path, 而此path即為s到v的最短路徑.

- 最短路徑的證明在上次的slides最後幾頁

Depth-first Search

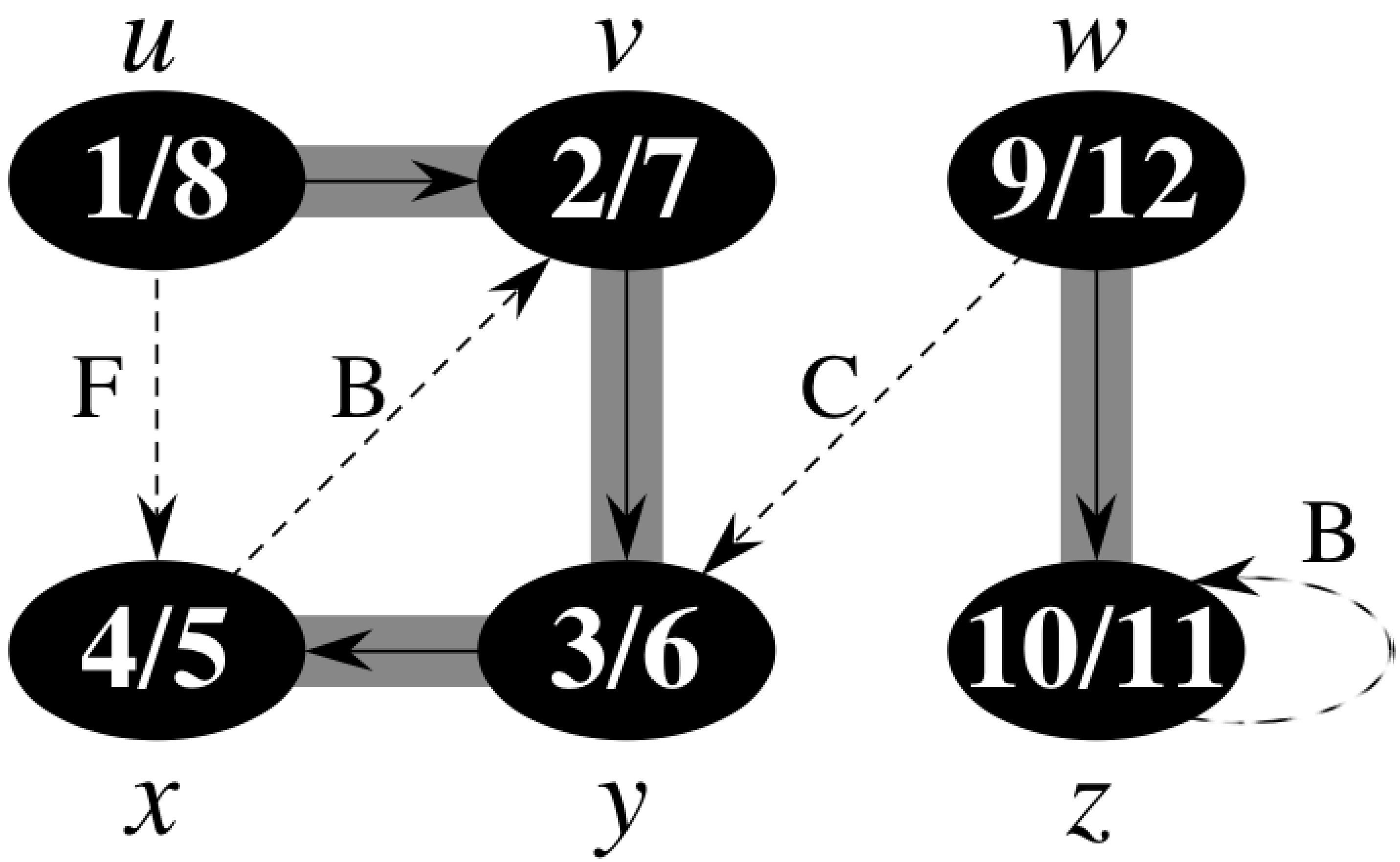
- 當可能的時候, 就往更深的地方找去 → Depth-first

- 和Breadth-first Search不同的地方:
- 會長出一個forest (多棵樹)
- 會記錄timestamp: 開始找到的時間(變成灰色)和完成所有和它相鄰的vertex的時間(變成黑色)

使用的structure

- 每一個vertex u有以下的structure:
- u.color: 紀錄目前這個vertex的狀態
  - WHITE: 這個vertex還沒被discover
  - GRAY: 這個vertex被discover了, 但是和它相連的vertex還沒都被discover
  - BLACK: 這個vertex及和它相連的vertex都已經被discover了
- u.pi: 紀錄它的前一個vertex (祖先) 是誰
- u.d: discover的時間 (從WHITE→GRAY的時間)
- u.f: finish的時間 (從GRAY→BLACK的時間)

- 時間(timestamp)總共會從1跑到2$|V|$, 因為每個vertex discover或finish會各增加一次timestamp.



如果邊的排列方式(Adjacency List)不同, 很可能會造成DFS出來的結果不同

試試看, 如果<u,x>比<u,v>先被走過, 請問會變怎麼樣?

Pseudo-Code

```
DFS(G)
for each vertex u ∈ G.V
  u.color=WHITE
  u.pi=NIL
time=0
for each vertex u ∈ G.V
      if u.color==WHITE
            DFS-VISIT(G,u)
```

```
DFS-VISIT(G,u)
time=time+1
u.d=time
u.color=GRAY
for each v ∈G.Adj[u]
      if v.color==WHITE
            v.pi=u
            DFS-VISIT(G,v)
u.color=BLACK
time=time+1
u.f=time
```

執行時間

```
DFS(G)
for each vertex u ∈ G.V
  u.color=WHITE
  u.pi=NIL
time=0
for each vertex u ∈ G.V
      if u.color==WHITE
            DFS-VISIT(G,u)
```

$\Theta(V)$

$\Theta(V)$

執行時間

總合起來: $\Theta(V+E)$

```
DFS-VISIT(G,u)
time=time+1
u.d=time
u.color=GRAY
for each v ∈G.Adj[u]
        if v.color==WHITE
            v.pi=u
            DFS-VISIT(G,v)
u.color=BLACK
time=time+1
u.f=time
```

$\Theta(V)$

DFS-Visit會執行|V|次

這個部分對每個vertex v會執行|Adj[v]|次
(vertex v的adjacency list長度)

$\Theta(E)$

$\Theta(V)$

把所有vertex的adjacency list長度加起來=|E|

動腦時間

- 如果我們改用Adjacency Matrix的話, 執行時間會變怎麼樣呢?

```
DFS-VISIT(G,u)
time=time+1
u.d=time
u.color=GRAY
for each v ∈G.Adj[u]
      if v.color==WHITE
            v.pi=u
            DFS-VISIT(G,v)
u.color=BLACK
time=time+1
u.f=time
```

??

Depth-first Forest

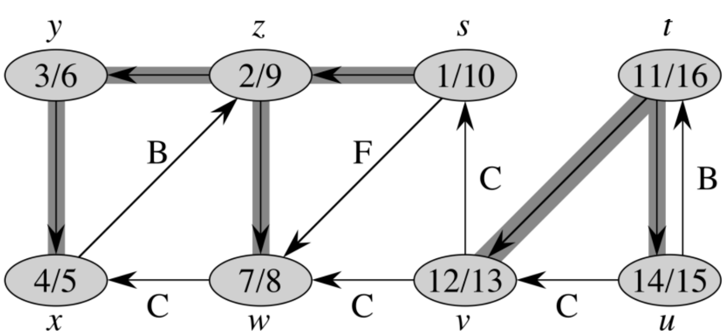
- 類似於breadth-first search, depth-first search也可以產生predecessor subgraph:
- $G_\pi = (V, E_\pi)$
- $E_\pi = \{(v.\pi, v): v \in V \; and \; v.\pi \neq NIL\}$.

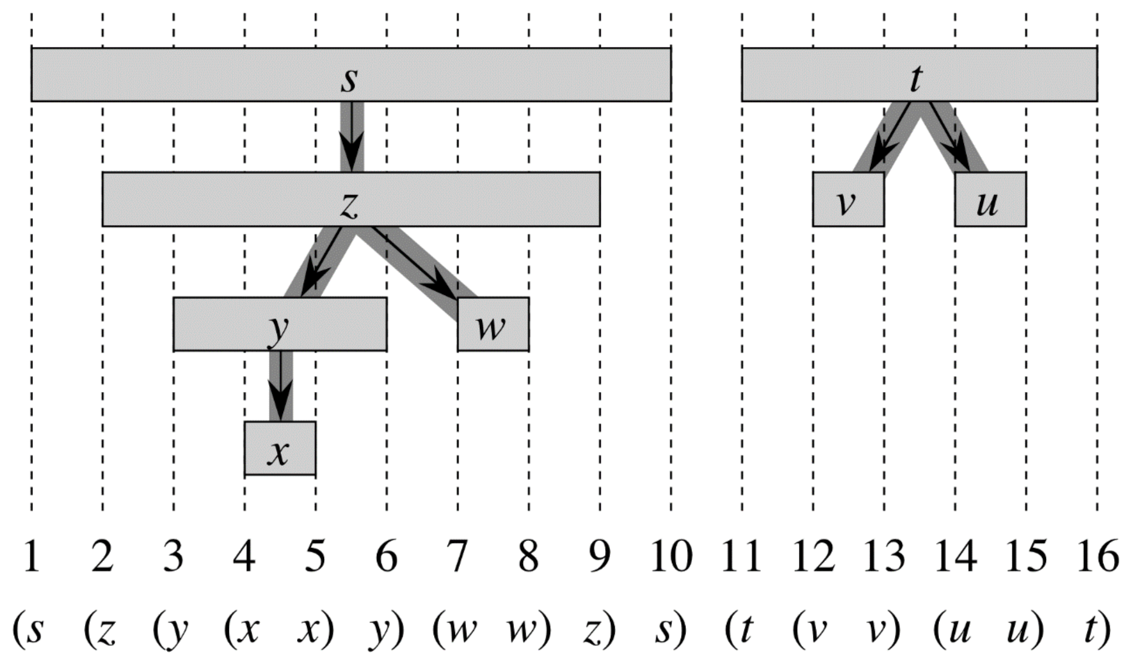
- 而Predecessor subgraph正好可以產生一個由多棵depth-first tree組成的 depth-first forest.

- 在E_π裡面的edge叫做tree edge.

Depth-first Search的括號結構性質

- Parenthesis structure: (括號結構)
- 如果我們用 (代表 vertex u的discovery被找到
- 用)代表 vertex u 的結束
- 則整個dfs的尋找歷史會使得vertex們的左括號和右括號都
  會不是互相包含
  就是相互分離





Reading assignment:
證明請見Cormen課本p.607-608

白路(White-path) 性質

從括號結構性質可得:
在depth-first forest of G裡面, v是u的子孫 iff u.d<v.d<v.f<u.f.

在G的depth-search forest裡面:

v是u的子孫 ⟺ 設定u.d的時候, u到v有一條全白的路徑

- (➔) :
- 如果v=u的話, 那設定u.d的時候, u還是白的.
- 如果v真的是u的子孫的話, u.d<v.d (v discover的時間比較晚)
- 此時v一定還是白的 (才在設定u.d而已)
- 既然v可以是任何u的子孫的話, 表示在u到v的路上(都是u的子孫)都也應該是白的

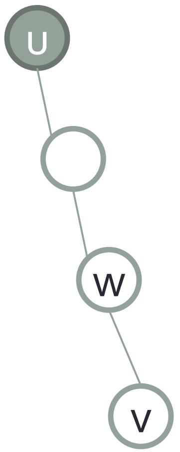
白路(White-path) 性質

從括號結構性質可得:
在depth-first forest of G裡面, v是u的子孫 iff u.d<v.d<v.f<u.f.

在G的depth-search forest裡面:

v是u的子孫 ⟺ 設定u.d的時候, u到v有一條全白的路徑

- (🡐) :
- 假設在u.d的時候, 有一條從u到v的全白路徑, 但是v不是u的子孫.
- 假設在白路徑上, 每個v之外的node, 都變成u的子孫.
  (意思其實就是取v使得他是u->v白路徑上第一個不是u子孫的node)
- 取w為v的u→v路徑上的前一個 (u和w有可能是同一個點)
- (1)由括號性質得$w.f \le u.f$
- (2)既然有白路徑, 所以$u.d < v.d$
- (3)v會在w結束前被找到(因為v和w有一條edge) 所以$v.d < w.f$
- 合併以上$u.d < v.d < w.f \le u.f$ → v.d被包含在 u.d和u.f中間了
- 由此可見, [v.d v.f] 應該為包含在 [u.d, u.f]裡面的狀況
- v為u之子孫, 矛盾



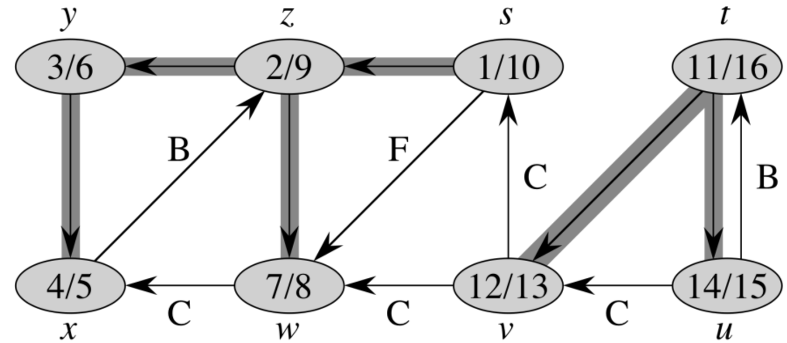
Depth-first forest 中的edge

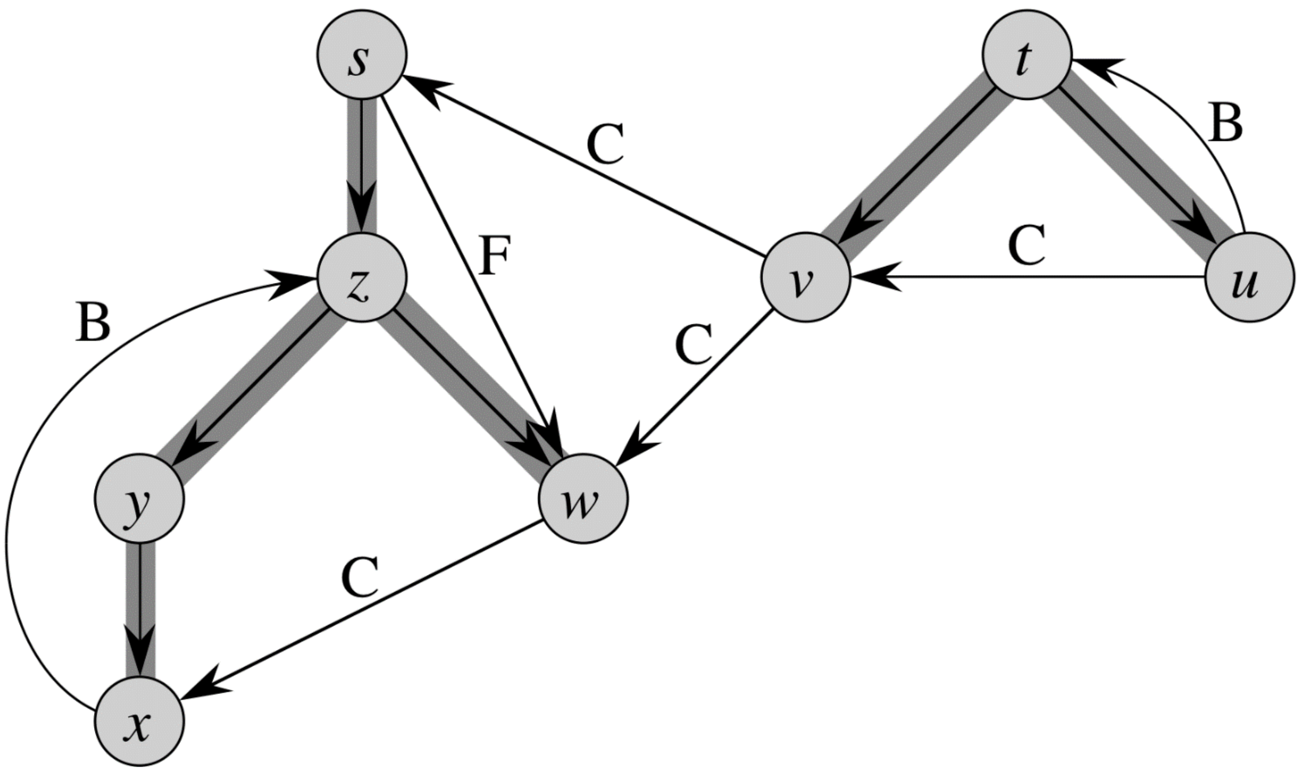
- 有四種:

1. Tree edge: 在depth-first forest裡面的邊叫做tree edge. 如果v是經由(u,v) discover的, 那(u,v)就是tree edge.
2. Back edge: 連接u到它的祖先v的邊(u,v)叫做back edge. Self-loop也算做是back edge的一種.
3. Forward edge: 連接u到它的子孫v的nontree edge (u,v).
4. Cross edge: 所有其他的edge. 可以是連接同一棵depth-first tree的邊, 或者是連接不同depth-first tree的邊.

例子


我是栗子


y
z
s
t
3/6
2/9
1/10
11/16
B
F
C
B
4/5
7/8
12/13
14/15
C
C
C
x
w
v
u


s
t
C
B
F
C
z
v
u
B
C
y
w
C
x

如何分辨是什麼邊呢?

- 當我們第一次碰到edge (u,v)的時候, v的顏色告訴我們它是什麼邊:

1. WHITE → tree edge
2. GRAY → back edge
3. BLACK → forward 或 cross edge
    1. u.d<v.d的話就是一條forward edge
    2. u.d>v.d的話就是一條cross edge

- 在undirected graph的depth-first forest 裡面沒有forward edge or cross edge. 想想看為什麼?

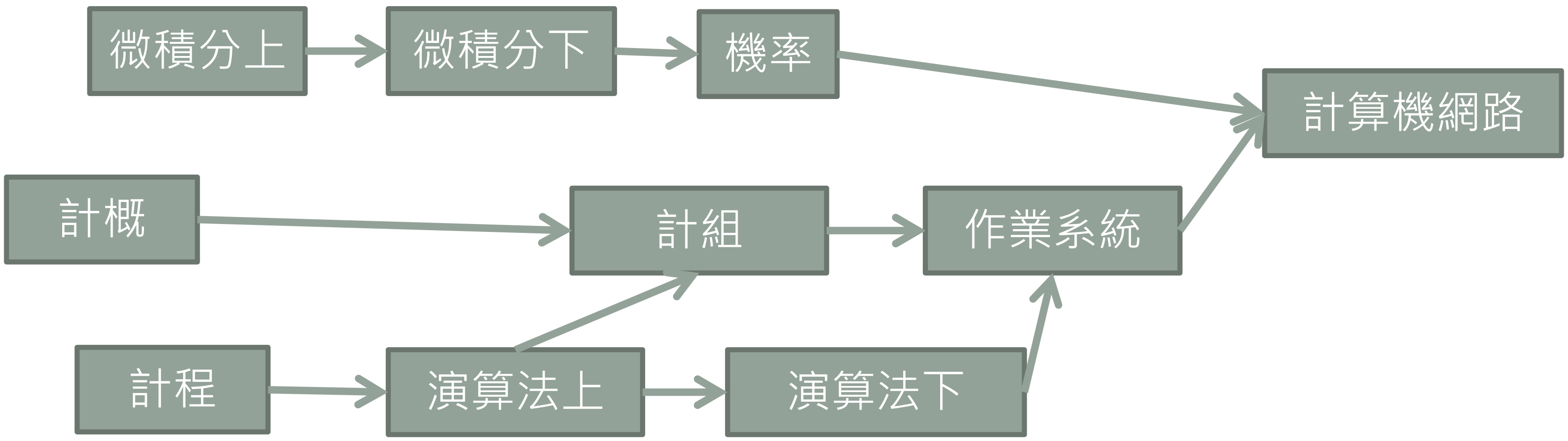
萬用DFS

- 可以用來幹嘛呢?
- 1. 看某graph G是不是connected
- 複習: 什麼是connected graph
- 怎麼做?

- 2. 列出所有的connected component
- 複習: 什麼是connected component
- 怎麼做?

- 後面還有

Topological Sort

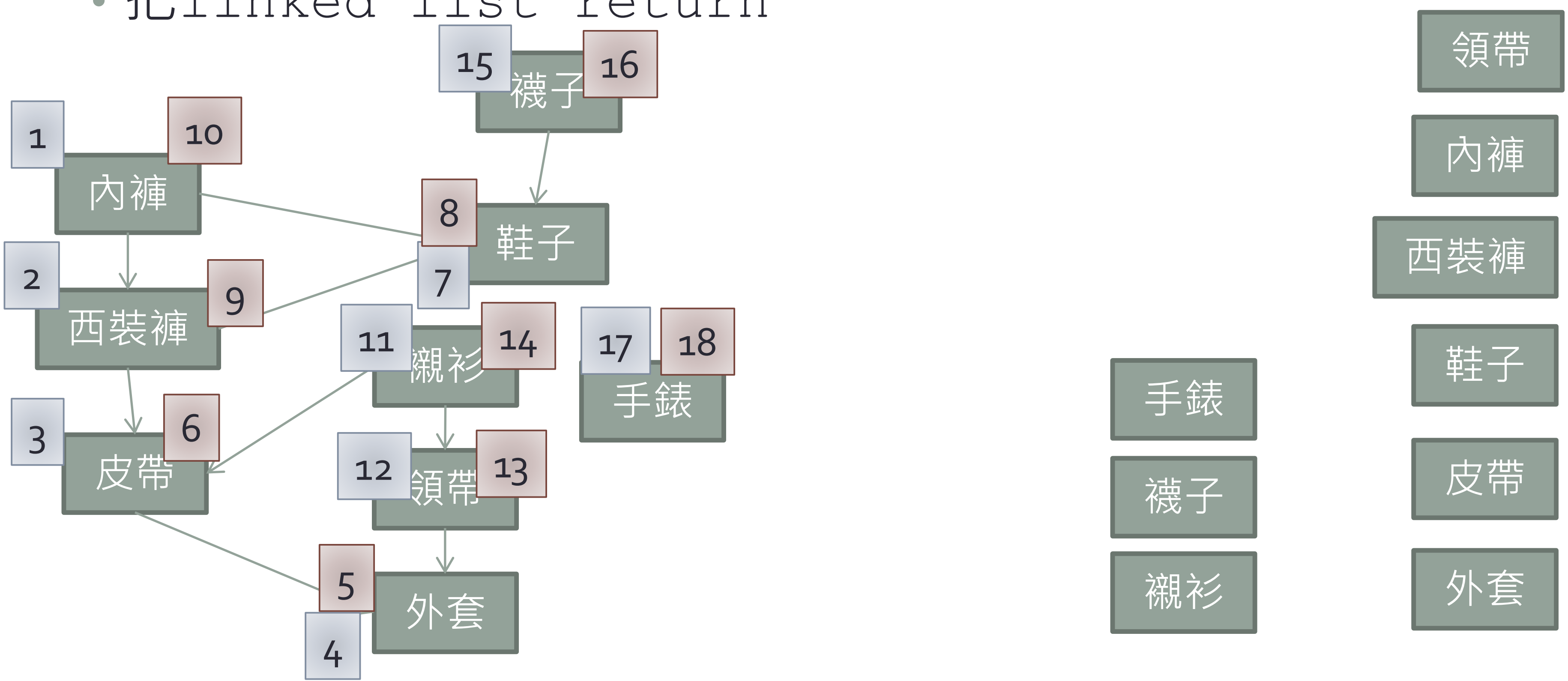


- 什麼時候可以修j課程呢? 修完所有edge<i,j>中i課程以後
- 所有的”activity”都是發生在vertex上
- 所以又稱為activity-on-vertex (AOV) network
- 問: 如何找到一種修課順序呢?
- 此順序又稱為topological order
- 要找topological order的graph必須是directed acyclic graph (DAG)

Topological Sort

$\boxed{\Theta(V+E)}$

- call DFS(G) 算出每一個vertex v的v.f
- 每個vertex finish的時候，就把它放到一個linked list的最前面
- 把linked list return



證明topological sort正確

- Lemma 22.11: G是acyclic iff G的depth-first search沒有back edges
- 證明:
- (➔)假設有back edge <u,v>, 那麼v在depth-first forest裡面應該是u的祖先
- 可是這樣的話表示可以用另外一條沿著tree edges從v走到u, 這樣就有 v→u→v的cycle. 矛盾. 所以應該沒有back edges.
- (🡐)假設有一個cycle c在G中.
- v是 c裡面第一個被discover的vertex. <u,v>是c裡面某一條邊.
- v.d的時候, v→u都是白色vertex, 因此根據白路徑定理, u會變成v的子孫, 因此 <u,v>是back edge. 矛盾
- 因此沒有cycle.

證明topological sorting正確

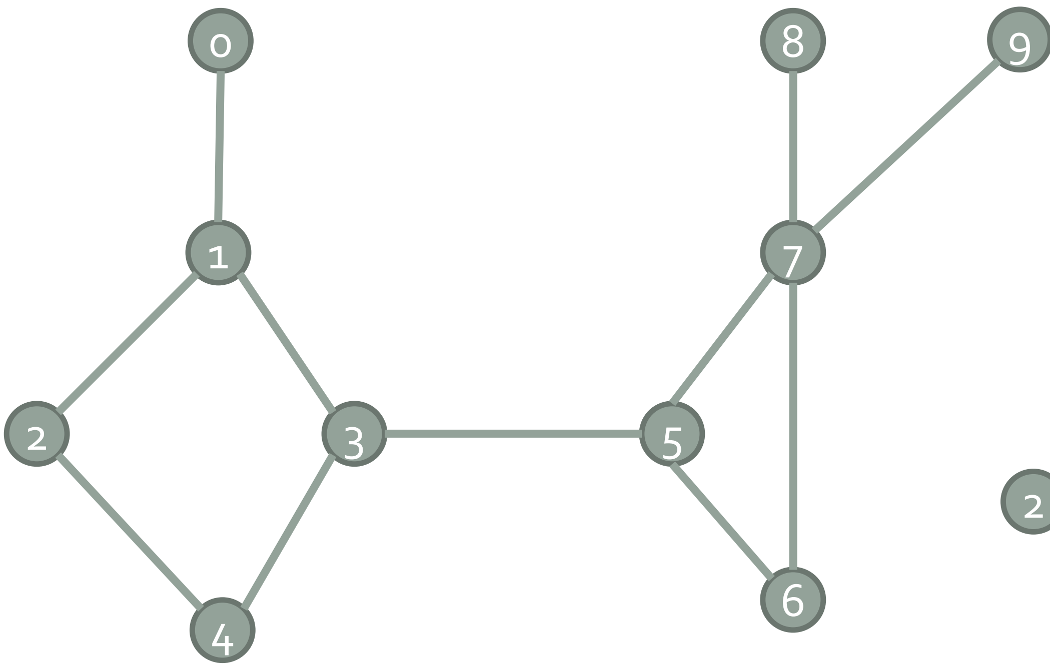
- 證明:
- 假設我們在G這個DAG上跑DFS
- 那麼對任何兩vertex u,v, 如果G裡面有<u,v>, 則v.f<u.f. 說明如下:
- 某個邊<u,v>被DFS經過的時候, v不可以是灰色的, 因為這樣的話, v就應該會是u的祖先, 這樣的話就有back edge了 (根據前一頁的定理, 矛盾)
- 所以v只能是黑或白色.
- 如果v是白色的, 那v就是u的子孫, v.f<u.f
- 如果v是黑色的, 那v已完成且v.f已經被設定. 但是我們還在從u往外尋找, 還沒有對u.f指定值, 所以 v.f<u.f
- 所以對任何edge <u,v>, v.f<u.f都成立.
- v.f值越小的放越後面, 所以只要有<u,v>, u都會比v先做
- 證畢!

Biconnected Components相關名詞

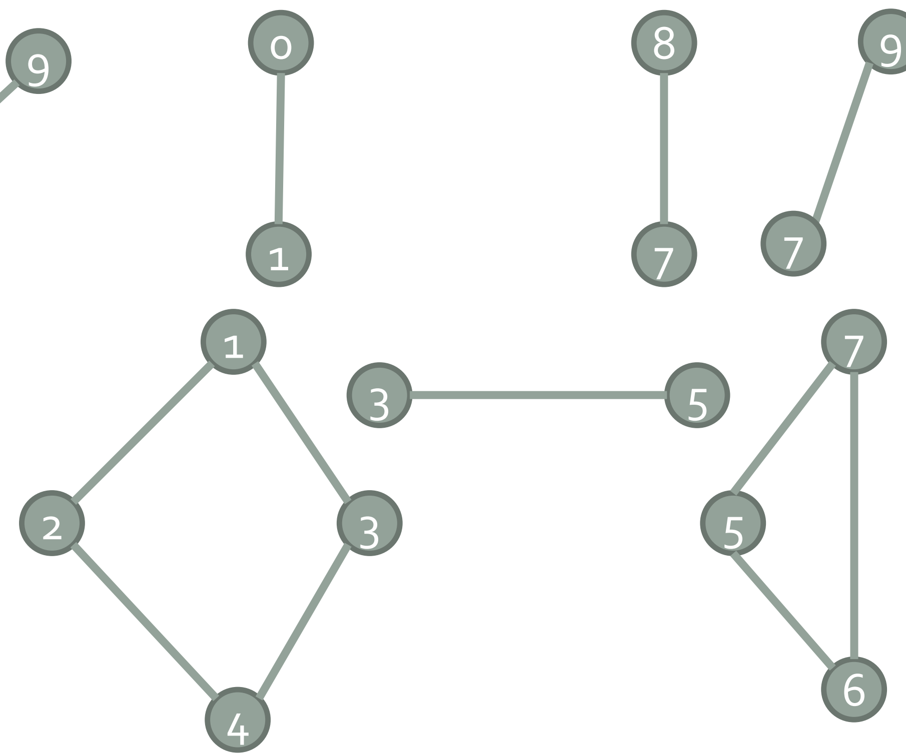
- Articulation point:
- 如果在connected graph G中的的一個vertex v被移除以後(包含v和所有incident在它上面的edge), 新的graph G'會變成有兩塊以上的connected components
- (複習: 什麼是 connected components)

- Biconnected graph: 沒有articulation point的graph
- Biconnected component: maximal biconnected subgraph
- 這邊maximal是說, 沒有一個可以包含它的subgraph是biconnected的

例子



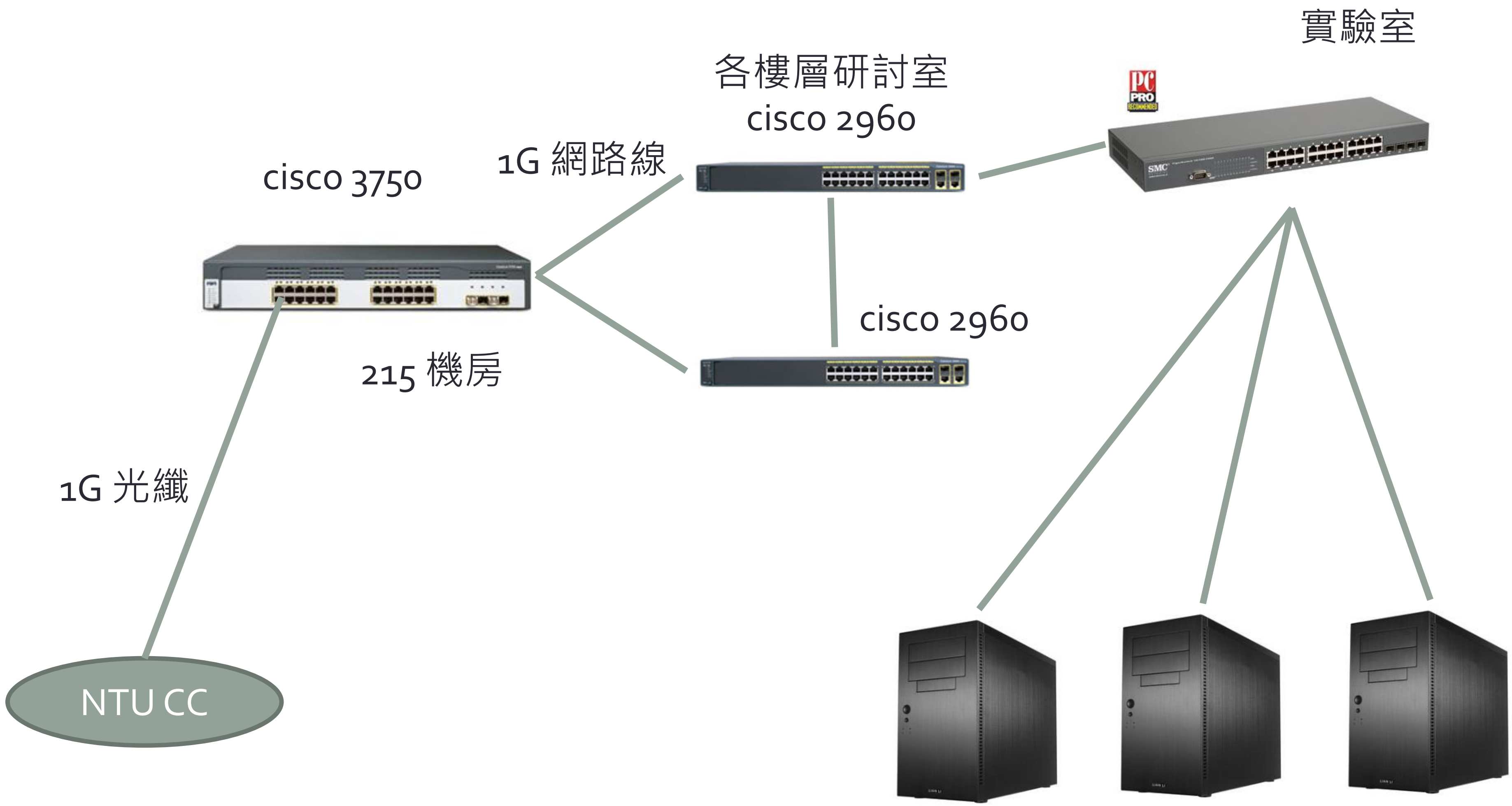
Biconnected components:



猜一猜哪邊是articulation point? (有沒有articulation point?)

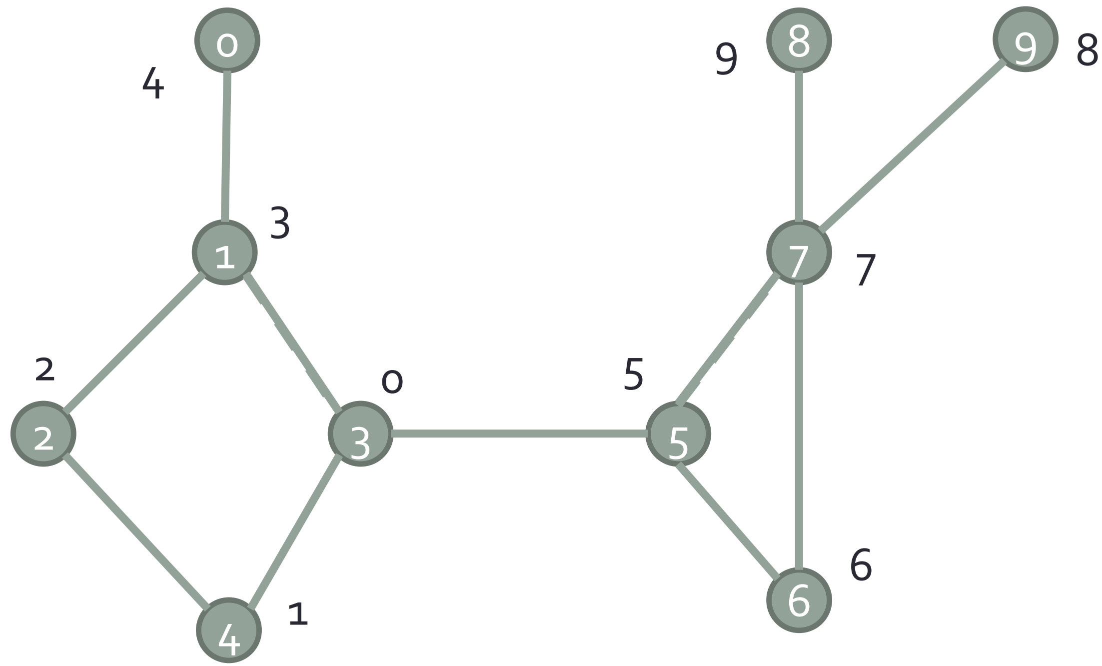
加了什麼邊可以讓它變成不是articulation point?

例子： 資訊系館網路



Where is the articulation point(s)?
How do we eliminate them?

DFS另一用途

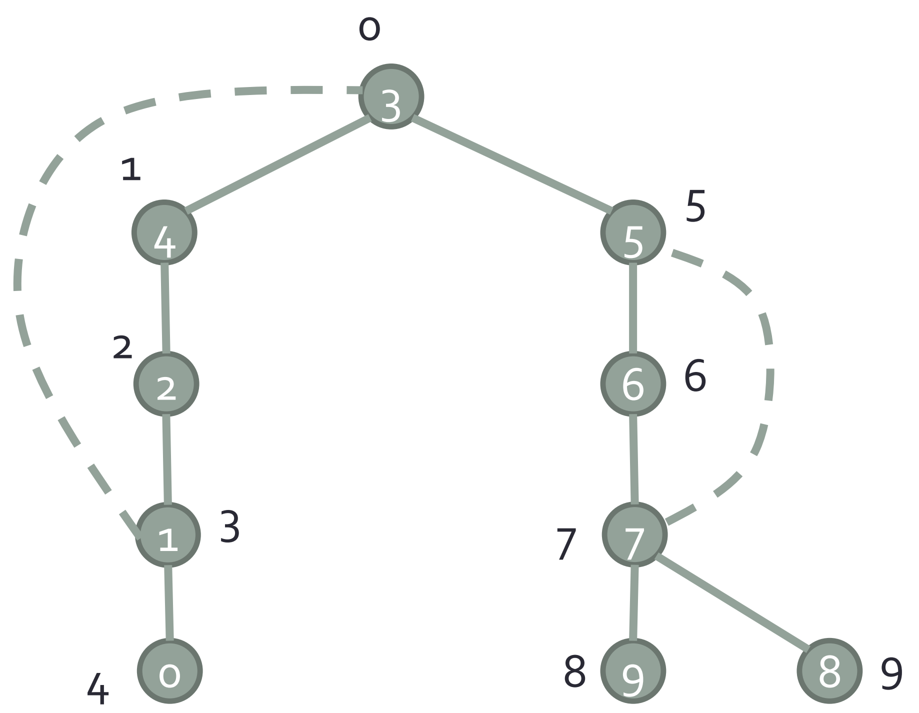


- 可以用來尋找articulation point & biconnected components

- 怎麼用呢? 首先先把graph做一次dfs, 並標上discover的順序
- 從哪個vertex開始不重要, edge順序也不重要
- (真的嗎? 自己驗證看看)

尋找articulation point

- 如果是root的話(開始做dfs的地方), 且有超過一個的children →是articulation point
- 如果不是root:
- 當有一個以上的小孩, 無法沿著它自己的後代及一條nontree edge (back edge)到達它的祖先的時候, 則為articulation point



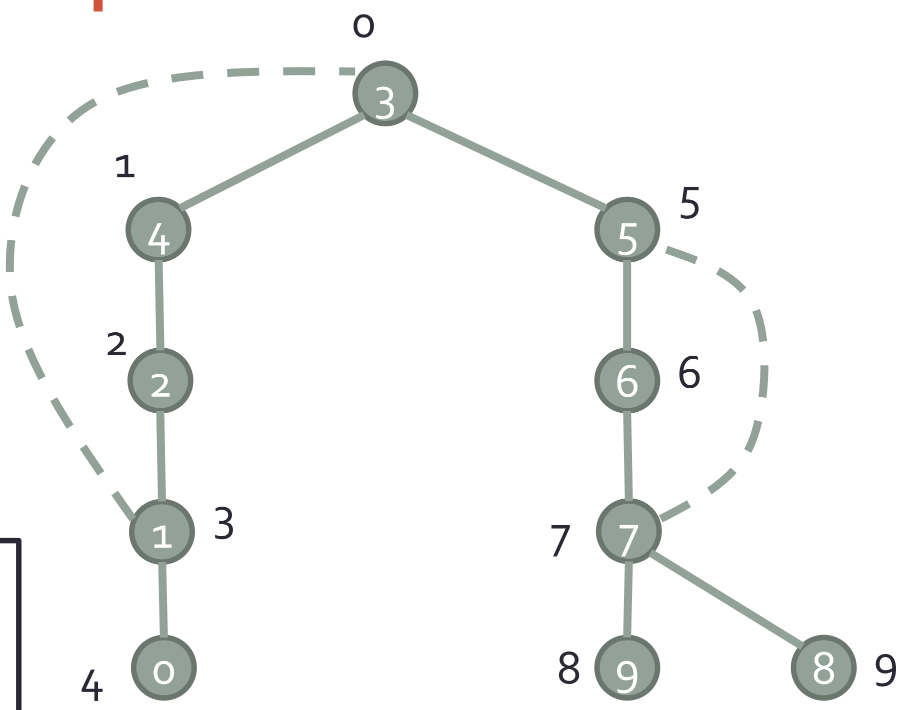
尋找articulation point

- 定義一些function來找articulation point

- $low(u) = \min\{u.d,$
  $\min\{low(w) | w \text{ is a child of } u\},$
  $\min\{w.d | (u, w) \text{ is a back edge}\}\}$

尋找articulation point

- Root有超過一個child即為articulation point
- 當某vertex v有任何一個child u的low(u)≥v.d時, 則v為articulation point



low(u)=min{u.d,
min{low(w)|w is a child of u},
min{w.d| (u,w) is a back edge}}

	0	1	2	3	4	5	6	7	8	9
v.d	4	3	2	0	1	5	6	7	9	8
low	4	0	0	0	0	5	5	5	9	8

Minimum Cost Spanning Tree

- 電路設計需要把N個電子零件的接點連接在一起
- 這些接點在不同的位置
- 要如何把它們通通連接起來呢? 需要有N-1條"電線", 每一條把兩個接點連接起來
- 怎麼樣才能使用最短的電線呢?

- 轉換: G=(V,E) 是undirected graph, V是接點, E所有可能的電線
- 每一個E (u,v)的weight w(u,v)代表所需使用的電線長度
- 須找出一個acyclic的subset $T \subseteq E$ 連接所有的vertex, 且使 $w(T) = \sum_{(u,v)\in T} w(u,v)$

Minimum Cost Spanning Tree

- T: acyclic且連接所有vertex, 所以為一棵tree且span到整個graph, 所以稱為spanning tree.

- 複習: Spanning tree須滿足那些條件?
- 1. 因為是tree, 所以沒有cycle
- 2. 因為是tree, 所以正好有n-1個edge

- 下面介紹三種使用greedy algorithm產生minimum cost spanning tree的方法

一般型Minimum Spanning Tree演算法

```
GENERIC-MST(G,w)
while A does not form a spanning tree
      find an edge (u,v) that is safe for A
      A=A∪{(u,v)}
return A
```

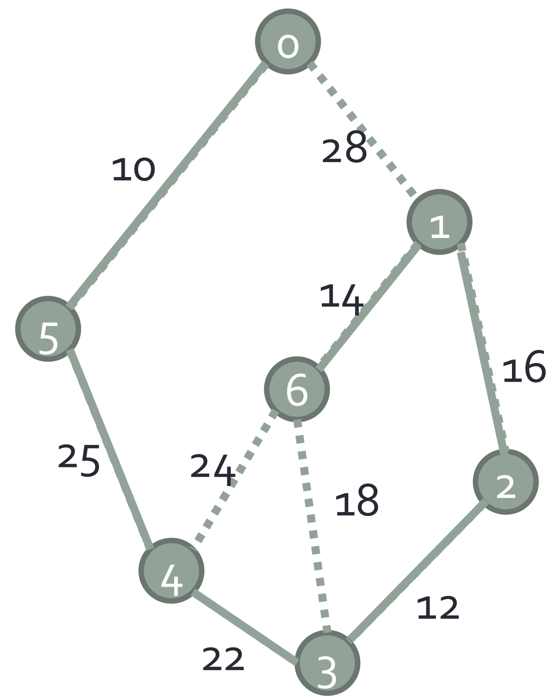
- A是最終的MST的edge的subset
- 隨時A都保持acyclic→隨時$G_A = (V, A)$ 都是一個forest
- 每次都選一個edge, 把 G_A 中兩棵tree連接起來 (因為不能出現cycle)
- 最後共選|V|-1條邊
- 下面三個algorithm的前兩個都是用這個方法
- Reading Assignment: 課本23.1

Kruskal’s algorithm

- 這個方法是我覺得最直觀的方法.

管理set的一些工具function:
MAKE-SET(v): 創造一個set
FIND-SET(v): 找到這個set的頭頭(id)
UNION(u,v): 把兩個set合併起來

```
MST-KRUSKAL(G,w)
A={}
for each vertex v ∈ G.V
     MAKE-SET(v)
sort G.E by w
for each edge (u,v) ∈ G.E
     if FIND-SET(u)≠FIND-SET(v)
          A=A∪{(u,v)}
          UNION(u,v)
return A
```



管理set的資料結構相關內容請見課本21章

Kruskal's algorithm

```
MST-KRUSKAL(G,w)
A={ }
for each vertex v ∈ G.V
      MAKE-SET(v)
sort G.E by w
for each edge (u,v) ∈ G.E
      if FIND-SET(u)≠FIND-SET(v)
            A=A∪{(u,v)}
            UNION(u,v)
return A
```

管理set的一些工具function:
MAKE-SET(v), FIND-SET(v), & UNION(u,v)
執行共m個operation的時候(n個item)
則執行時間為$O(m\alpha(n))$

$|V|$次

$O(E\log E)$

$O(E)$次

$O(E)$次

$O((V+E)\alpha(V))$

$|E| \geq |V|-1$ since G is connected

$\alpha(V)=O(\log V)=O(\log E)$

$O(E\alpha(V))$

$|E|<|V|^2$, 所以 $\log|E|=O(\log|V|)$

$O(E\log V)$

證明題

- 證明Kruskal's algorithm會產生出minimum cost spanning tree.
- (1) 證明當某graph有spanning tree的時候, Kruskal's algorithm會產生出spanning tree.
- (2) 證明這個產生出來的spanning tree一定是cost最小的.

- 證明(1):
- 什麼時候某graph一定有spanning tree呢?
- →原本是connected的
- Algorithm什麼時候會停下來呢?
- (1) 當T裡面已經有n-1個edge了, 且正好每個edge都處理完畢 (成功, 不管它)
- (2) T裡面還沒有n-1個edge, 但是每個edge都處理完畢, 而有些node沒有連接到 (我們的algorithm會不會造成這樣的情形呢?)

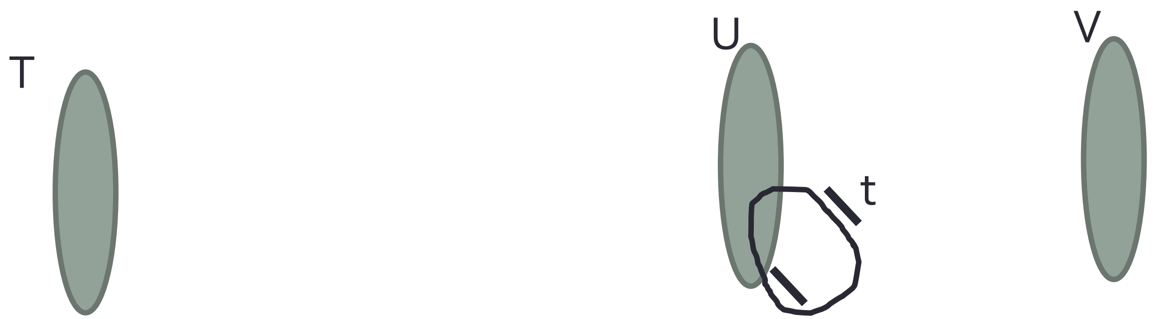
- 但是我們的程式只會把造成cycle的edge丟掉, 當把造成cycle的edge丟掉的時候, 不會因此讓某個node在T裡面沒有跟其他vertex connected.
- 所以不會造成(2)的情形

證明題

- 證明(2)
- 假設T是用我們的algorithm做出來的spanning tree
- 假設U是某一個minimum cost spanning tree (可能有很多個, 其中一個)
- 既然都是spanning tree, T和U都有n-1個edge
- 有兩種情形:
- (1) T和U一模一樣, 則T就是minimum cost spanning tree (沒什麼好證的)
- (2)T和U不一樣. 則我們假設它們有k條edge不一樣, $k > 0$.

證明題

- 每次我們從T取出k條不一樣的edge中其中一條(此edge不在U中), 從cost最小的開始到cost最大的.
- 把這條edge(我們叫它t)加入U的時候, 會產生一個cycle在U中
- 這個cycle裡面, 一定有某一條edge不在T裡面, 我們叫它u (因為T沒有cycle). 我們把u從U拿掉, 這個新的spanning tree叫做V
- V的total cost就是 cost(U)-cost(u)+cost(t).



k條不在U中的, 其中最小的為t

證明題

- 但是cost(t)不能小於cost(u), 否則V就比U的cost少了(contradiction)
- cost(t)也不能大於cost(u),
- 不然當初我們做T的時候, 應該會先選到u, 但是因為它會造成cycle所以才不選它.
- 所以u和所有在T裡面cost跟cost(u)一樣大或者更小的edge會造成cycle.
- 但是剛剛既然我們先選到t (在T裡面不在U裡面最小的一個), 表示這些cost跟cost(u)一樣大或者更小的edge都在U裡面
- 表示u和這些edge都在U裡面, U會有cycle (contradiction)

證明題

- 搞了半天, 目前可以證明cost(t)=cost(u)
- 所以cost(V)=cost(U)
- 重複以上步驟, 可以最後變成 V=T 且 cost(V)=cost(T)=cost(U)
- 所以T也是minimum cost spanning tree

Today’s Reading Assignment

- Cormen 22.3 22.4
- Horowitz 6.2.5 (p. 286-291) (Articulation point & Bi-connected components)

- Minimum Spanning Tree的部分:
  - Cormen 23的內容比較困難, 有時間的同學可以挑戰看看!

下課時間


駝鳥型
ostrich
新奇度
睡覺以此姿勢來看，想必此人缺乏安全感
想睡又怕被抓包的矛盾心理下，
自然而然擺出這樣的姿態，
身體語言：你看不到我你看不到我

下課時間


死撐型
keep
新奇度 ★★★☆☆
昨晚熬夜讀書的好學生,
在昏昏欲睡之時, 仍然不肯接受睡魔的邀請
繼續努力聽課, 是好學生的模範
身體語言:尊師重道的好孩子