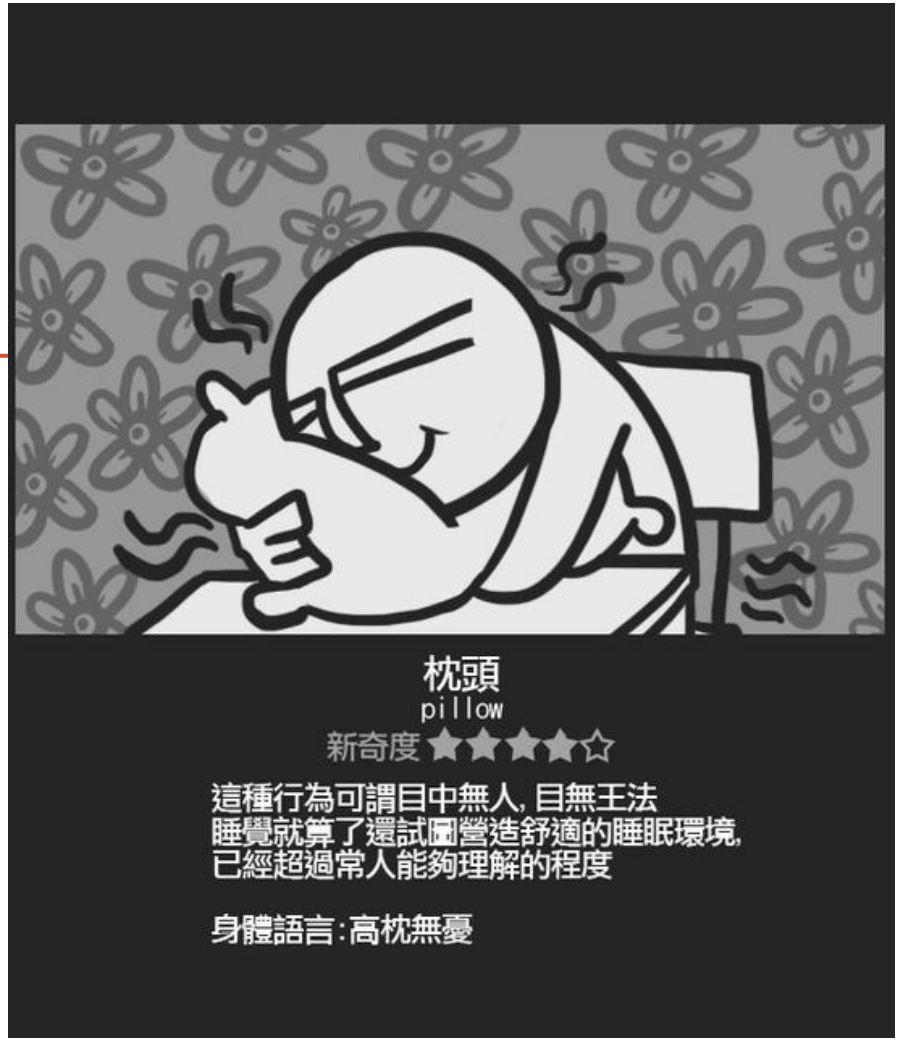
GRAPH

Michael Tsai

2011/11/25

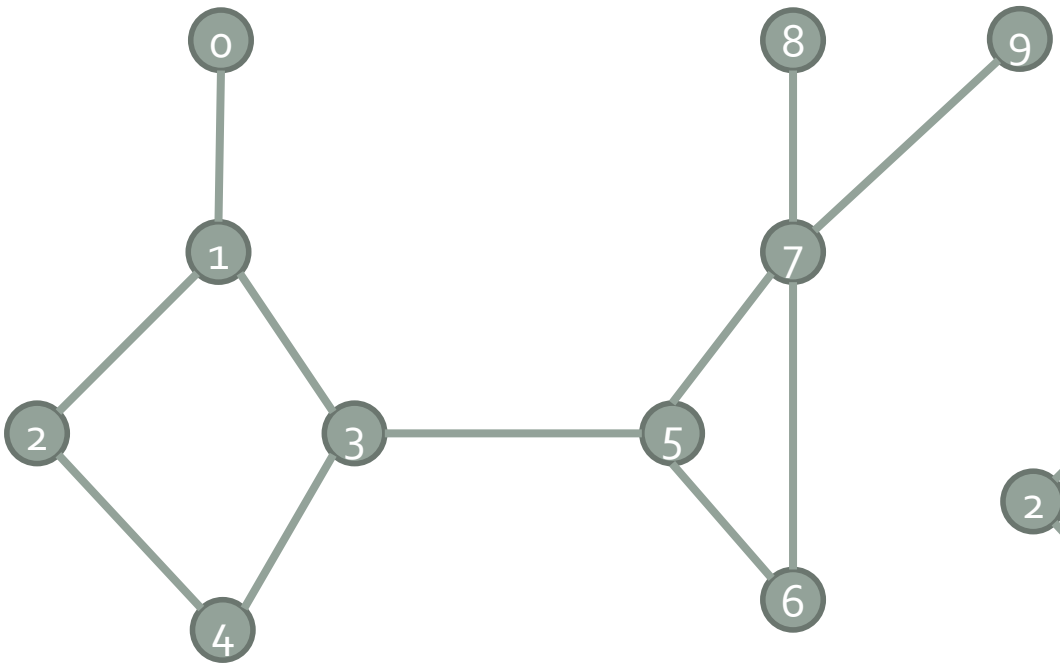


Biconnected Components相關名詞

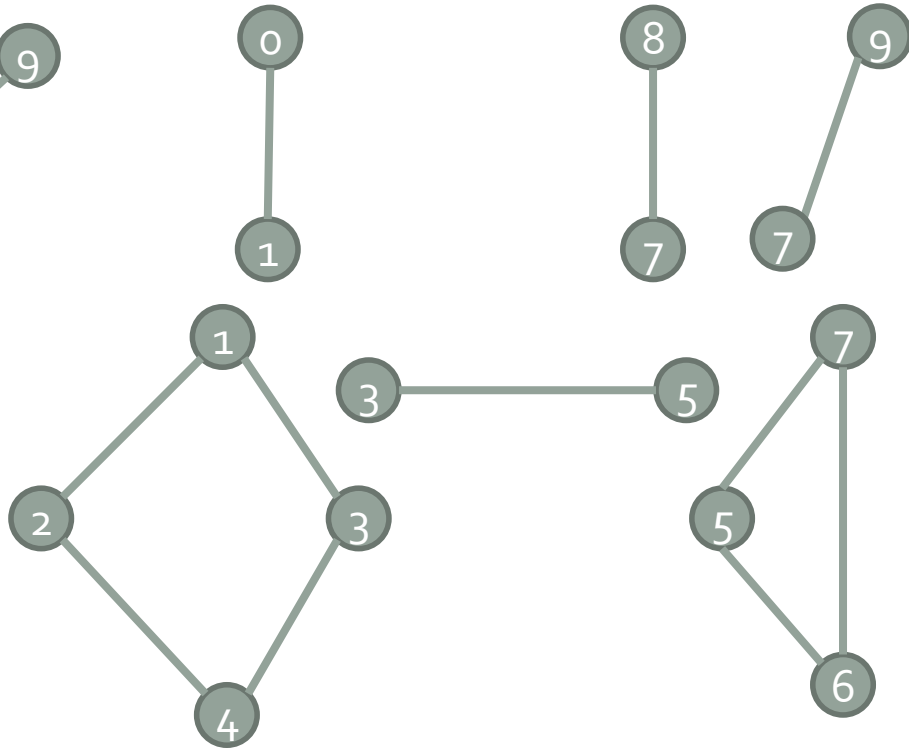
- Articulation point:
- 如果在connected graph G中的的一個vertex v被移除以後(包含v和所有incident在它上面的edge), 新的graph G'會變成有兩塊以上的connected components
- (複習: 什麼是 connected components)

- 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



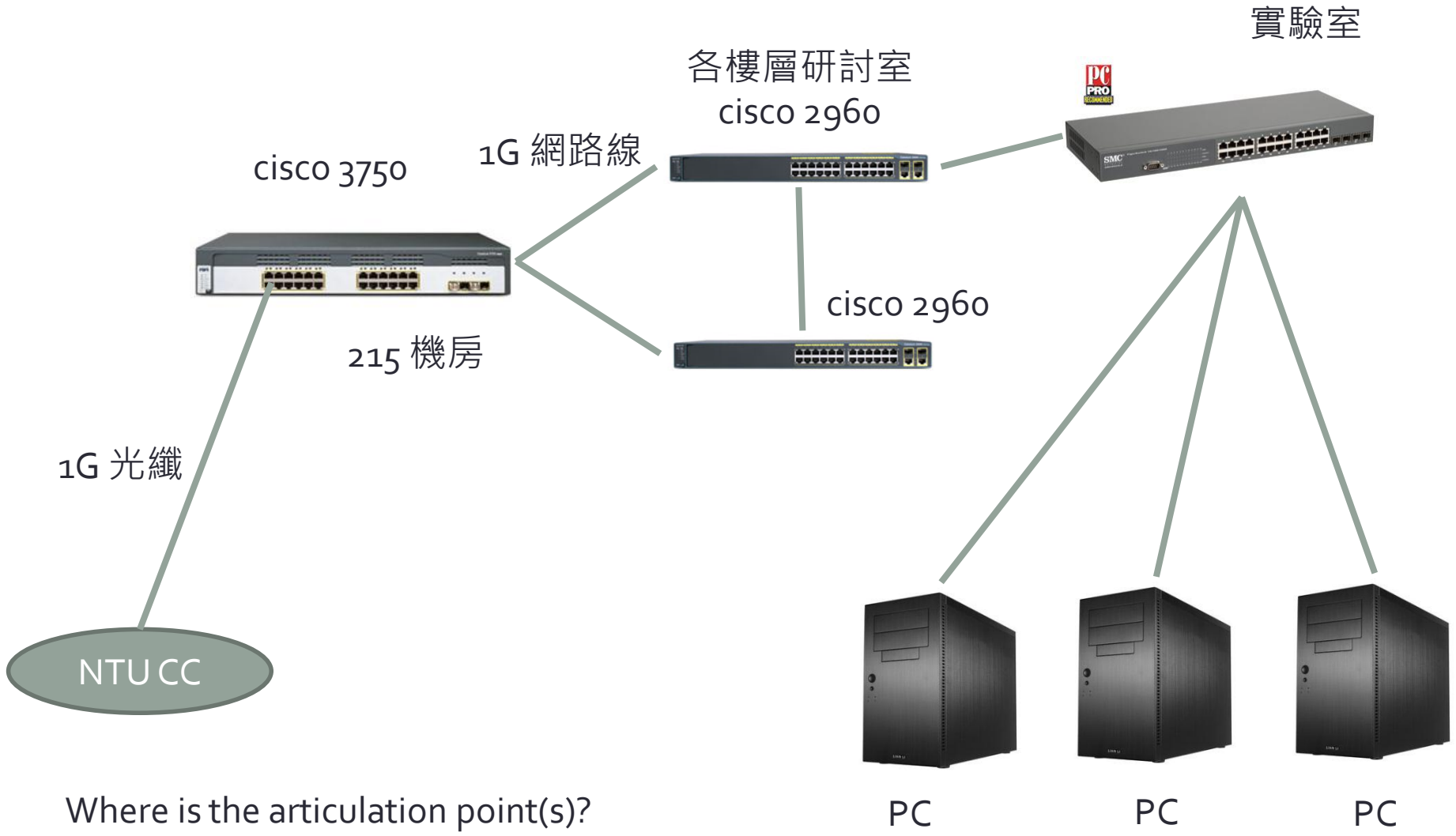
Biconnected components:



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

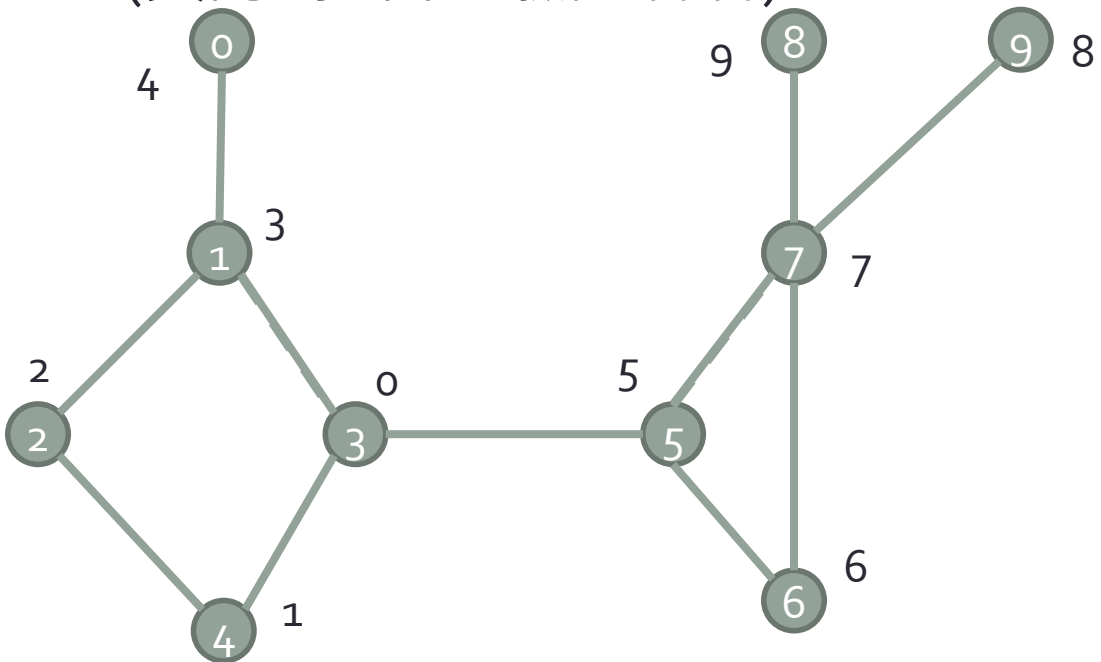


Where is the articulation point(s)?
How do we eliminate them?

DFS另一用途

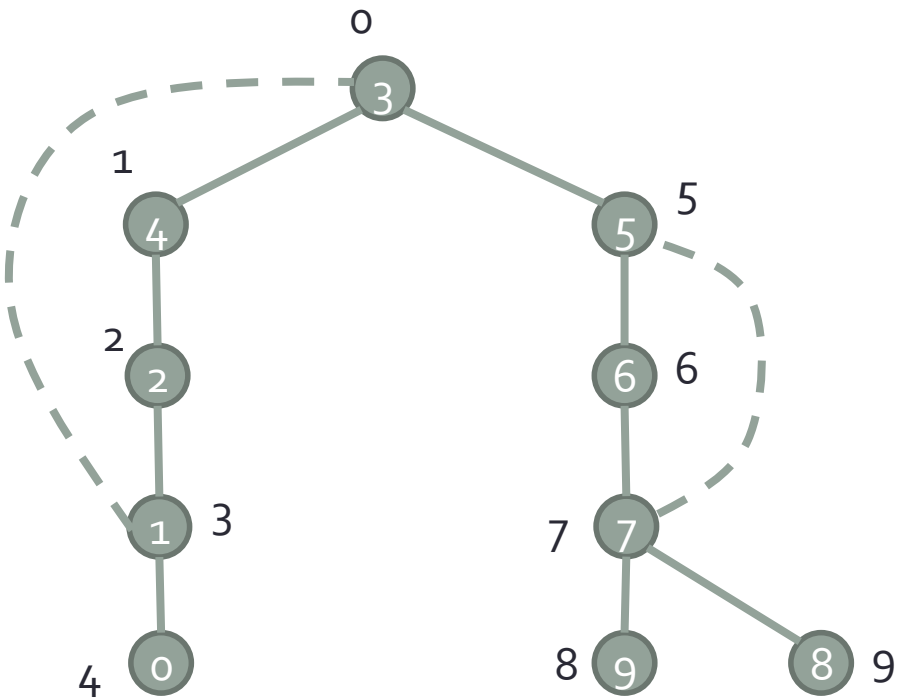
- 可以用來尋找articulation point & biconnected components

- 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
- back edge: 一條edge (u,v), u是v的祖先或者v是u的祖先.



尋找articulation point

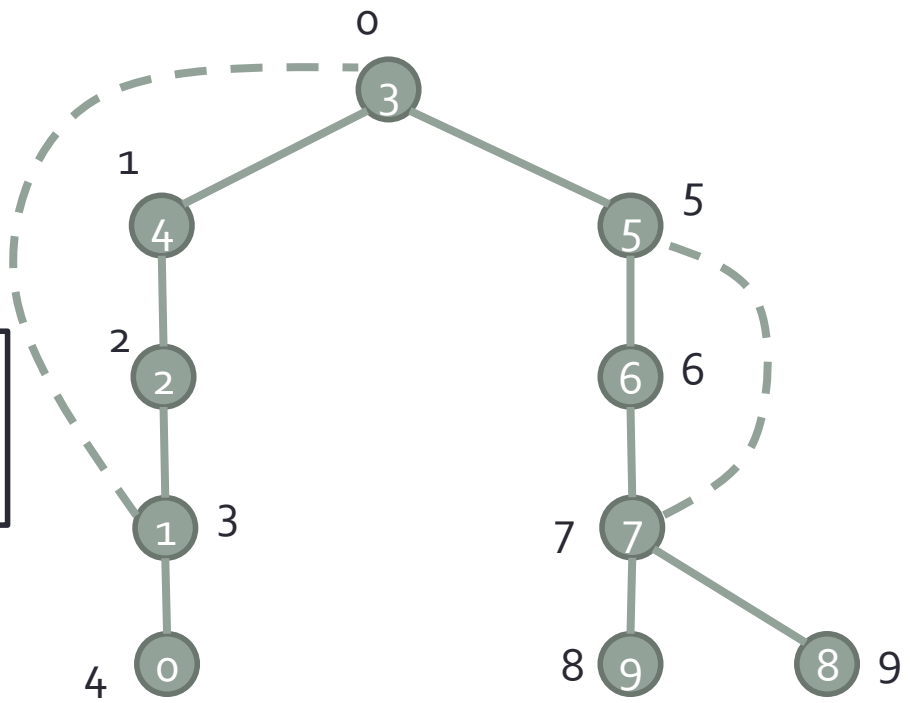
- 定義一些function來找articulation point

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

尋找articulation point

- 當某vertex v有任何一個child的low值>=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=AU{(u,v)}
return A
```

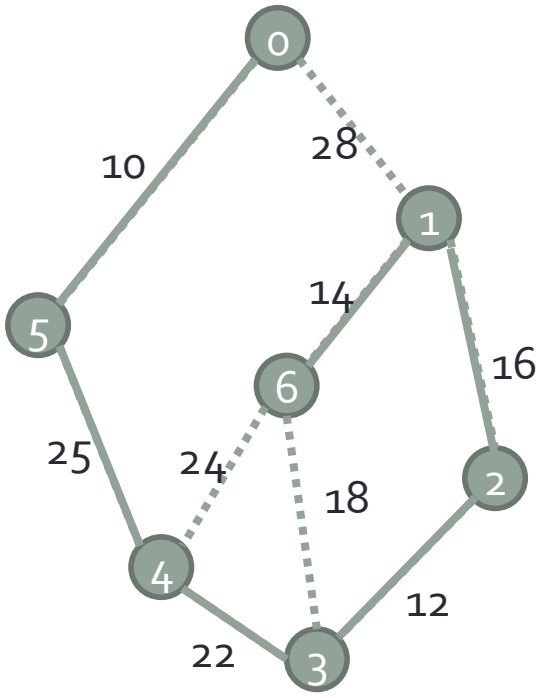
- A是最終的MST的edge的subset
- 隨時A都保持acyclic→隨時$G_A = (V, A)$ 都是一個forest
- 每次都選一個edge, 把 G_A 中兩棵tree連接起來 (因為不能出現cycle)
- 最後共選|V|-1條邊
- 下面三個algorithm的前兩個都是用這個方法
- Reading Assignment: 課本23.1

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?)

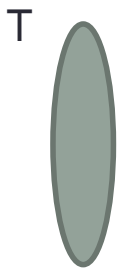
- 但是我們的程式只會把造成cycle的edge丟掉, 當把造成cycle的edge丟掉的時候, 不會因此讓某個node在T裡面沒有跟其他vertex connected.
- 所以不會造成(2)的情形

證明題

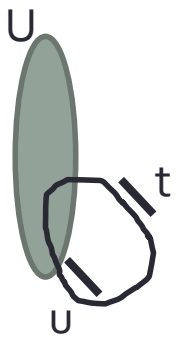
- 證明(2)
- 假設T是用我們的algorithm做出來的spanning tree
- 假設U是某一個minimum cost spanning tree (可能有很多個, 其中一個)
- 既然都是spanning tree, T和U都有n-1個edge
- 有兩種情形:
- (1) T和U一模一樣, 則T就是minimum cost spanning tree (沒什麼好證的)
- (2)T和U不一樣. 則我們假設它們有k條edge不一樣, $k \geq 0$.

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

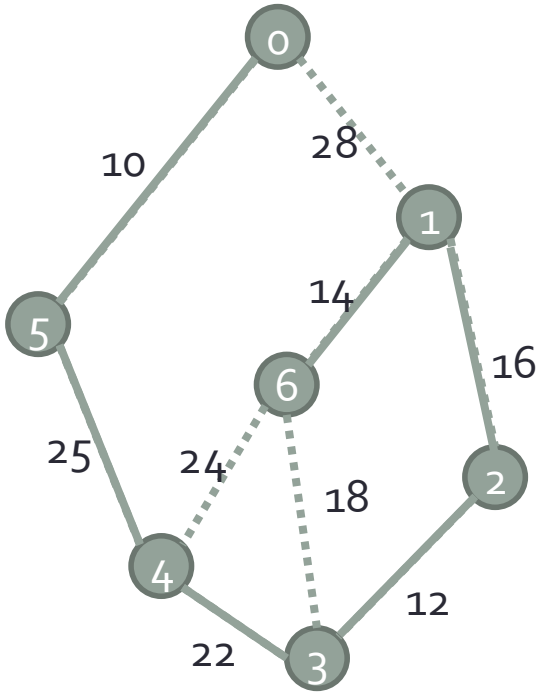
- 搞了半天, 目前可以證明cost(f)=cost(e)
- 所以cost(V)=cost(U)
- 重複以上步驟, 可以最後變成 V=T 且 cost(V)=cost(T)=cost(U)
- 所以T也是minimum cost spanning tree

Prim's Algorithm

u.key: 記錄到G_A最短的邊的weight
u.pi: 紀錄上面講的那條邊的另外一端是哪個vertex

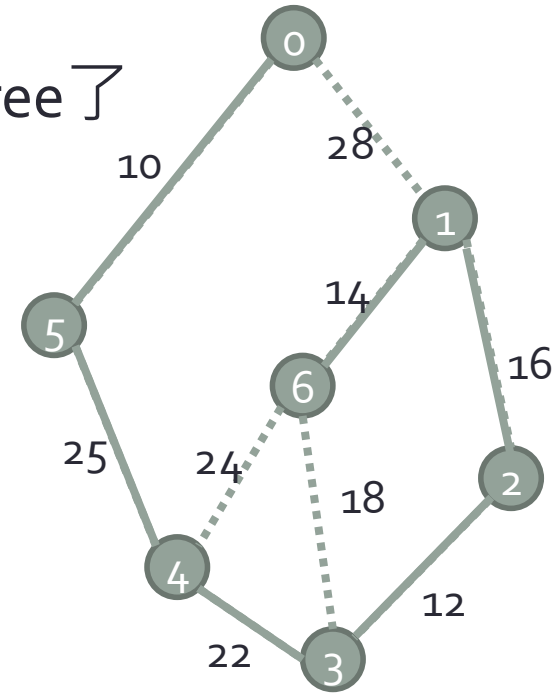
```
MST-PRIM(G,w,r)
for each u ∈ G.V
        u.key=∞
        u.pi=NIL
r.key=0
Q=G.V
while Q≠{}
        u=EXTRACT-MIN(Q)
        for each v ∈G.Adj[u]
                if v∈ Q and w(u,v)<v.key
                        v.pi=u
                        v.key=w(u,v)

A={(v,v.pi):v∈V-{r}}
```



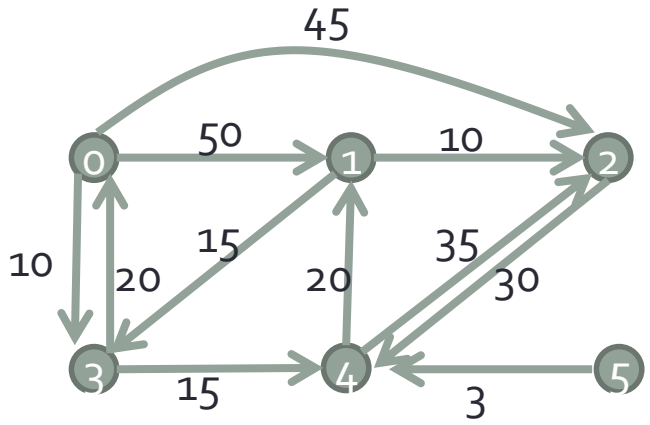
Sollin's algorithm

- 一開始每個vertex自己是一個forest
- 保持都是forest
- 每個stage, 每個forest都選擇一個該forest連到其他forest的edge中cost最小的一個
- 執行到沒有edge可以選了, 或者是變成tree了



Shortest paths

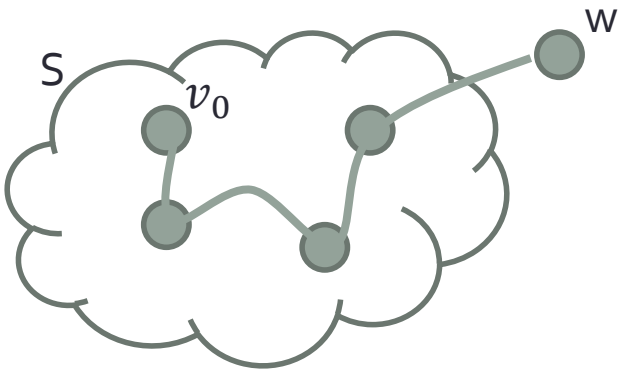
- 目標: 找出vertex u到graph G中任何一點的最短距離



TO VERTEX	PATH	LENGTH
3	0 3	10
4	0 3 4	25
1	0 3 4 1	45
2	0 2	45
5	No path	∞

Dijkstra's algorithm

- Set S裡面有一些已經加入的vertex (包括起始點v_0)
- w不在S中, 則有distance[w]紀錄從v_0經過S中的任意個vertex後, 最後到達w的這樣形式的最短路徑



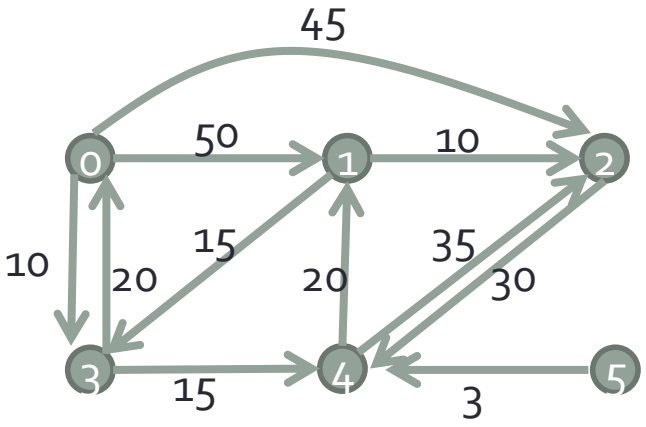
Dijkstra's algorithm

- 每次選擇u, 為distance[u]中最小的
- 選進S以後, v_0到u的shortest path就找到了
- 所以找到shortest path的順序會是由最短的找到最長的
- 把u選進去S以後, 就要把所有u的edge <u,w>都看一遍:
- if distance[w]>distance[u]+cost(<u,w>)
- distance[w]=distance[u]+cost(<u,w>)

小證明

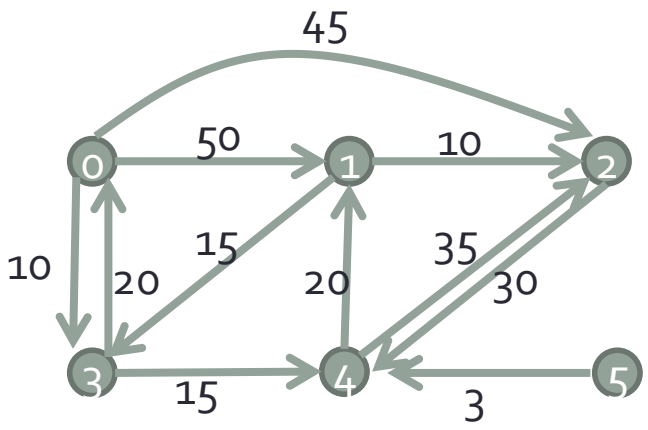
- 如果下一個選擇的vertex為u, 為什麼v_0到u的最短path會正好是前面的vertex都在S中, 只有u不在S中呢?
- 因為假如有path中有vertex w也不在S中, 那麼表示path中會有一段是v_0到w, 而且長度比較短 (因為是sub path)
- 可是這樣的話, 應該w之前應該就要被選過了 (應該要在S中) (contradiction)
- 所以v_0到u的最短path前面的vertex都在S中, 只有u不在S中

例子



selection	0	1	2	3	4	5
Initial	0	50	45	10	∞	∞

例子



selection	0	1	2	3	4	5
Initial	0	50	45	10	∞	∞
3	0	50	45	10	25	∞
4	0	45	45	10	25	∞
1	0	45	45	10	25	∞
2	0	45	45	10	25	∞

更多動畫例子

- http://www-b2.is.tokushima-u.ac.jp/~ikeda/suuri/dijkstra/DijkstraApp.shtml?demo1

- Time complexity?
- 答案: $O(|V|^2)$
- 每次都要看一次distance[], 共n個

Bellman-Ford algorithm

- 另外一種找到shortest path的algorithm
- 可以handle path cost是負的情形

- 為什麼Dijkastra 有問題?

- 如果下一個選擇的vertex為u, 為什麼v_0到u的最短path會正好是前面的vertex都在S中, 只有u不在S中呢?
- 因為假如有path中有vertex w也不在S中, 那麼表示path中會有一段是v_0到w, 而且長度比較短 (因為是sub path)
- 可是這樣的話, 應該w之前應該就要被選過了 (應該要在S中) (contradiction)
- 所以v_0到u的最短path前面的vertex都在S中, 只有u不在S中

假設cost可以是負的, 這就不一定正確了.

Bellman-Ford algorithm

- 定義 $dist^{l}[u]$為從v_0到u的最短路徑, 最多經過l條edge

- 一開始$dist^{1}[u]$為cost(<v_0,u>) (如果沒有edge則為無限大)
- 由$dist^{k-1}[]$算$dist^{k}[]$
- 最後算到$dist^{n-1}[u]$為最後shortest path解 (因為每個node最多走一遍, 不然就有cycle了)

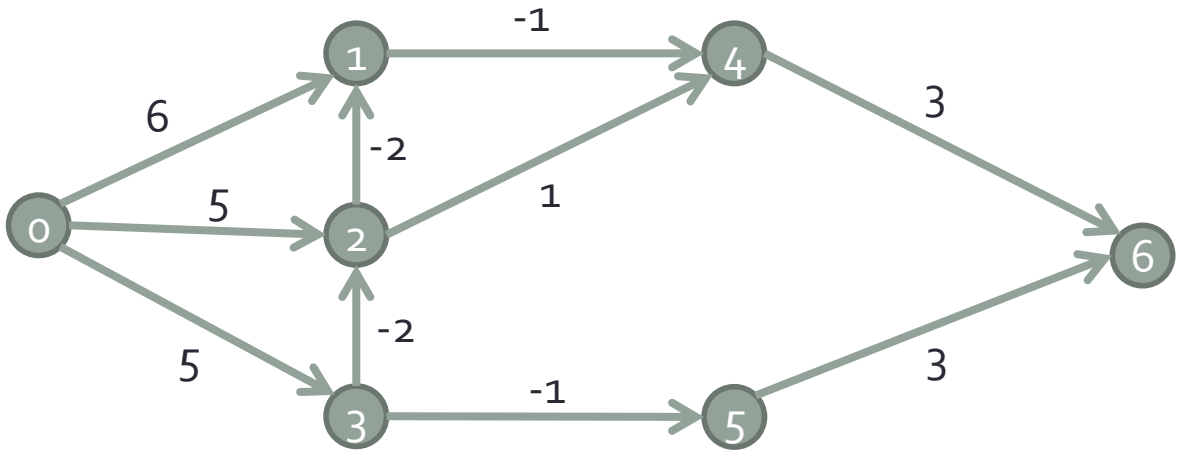
Bellman-Ford algorithm

- 由$dist^{k-1}[]$算$dist^{k}[]$
- 怎麼算呢?

- 有以下兩種可能性:
- 如果從v_0到u的使用最多k個edge的最短路徑其實使用了k-1或更少edge的話, 則$dist^{k}[u] = dist^{k-1}[u]$
- 如果從v_0到u的使用最多k個edge的最短路徑使用了k條edge的話, 則最短路徑可能為經過別的vertex i的shortest path (用了最多k-1條edges)後再走edge <i,u>.

- 綜合以上兩條規則:
- $dist^{k}[u] = \min\{dist^{k-1}[u], \min_{i}\{dist^{k-1}[i] + cost(\langle i,u\rangle)\}\}$

例子

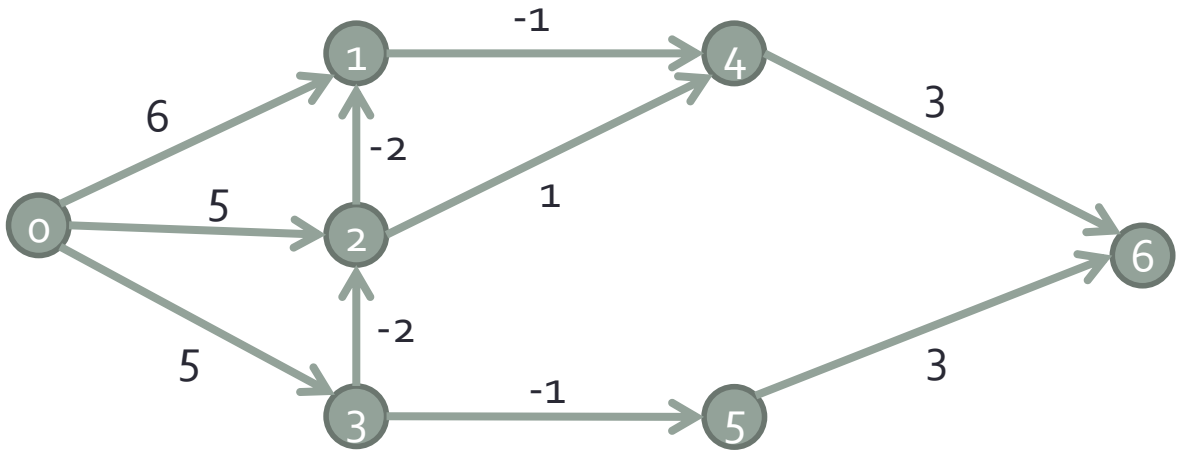


vertex 0到其他vertices的最短路徑:

k	1	2	3	4	5	6
1	6	5	5	∞	∞	∞
2						
3						
4						
5						
6						

其他例子: http://www.ibiblio.org/links/applets/appindex/graphtheory.html

例子



k	1	2	3	4	5	6
1	6	5	5	∞	∞	∞
2	3	3	5	5	4	∞
3	1	3	5	2	4	7
4	1	3	5	0	4	5
5	1	3	5	0	4	3
6	1	3	5	0	4	3

其他例子: http://www.ibiblio.org/links/applets/appindex/graphtheory.html

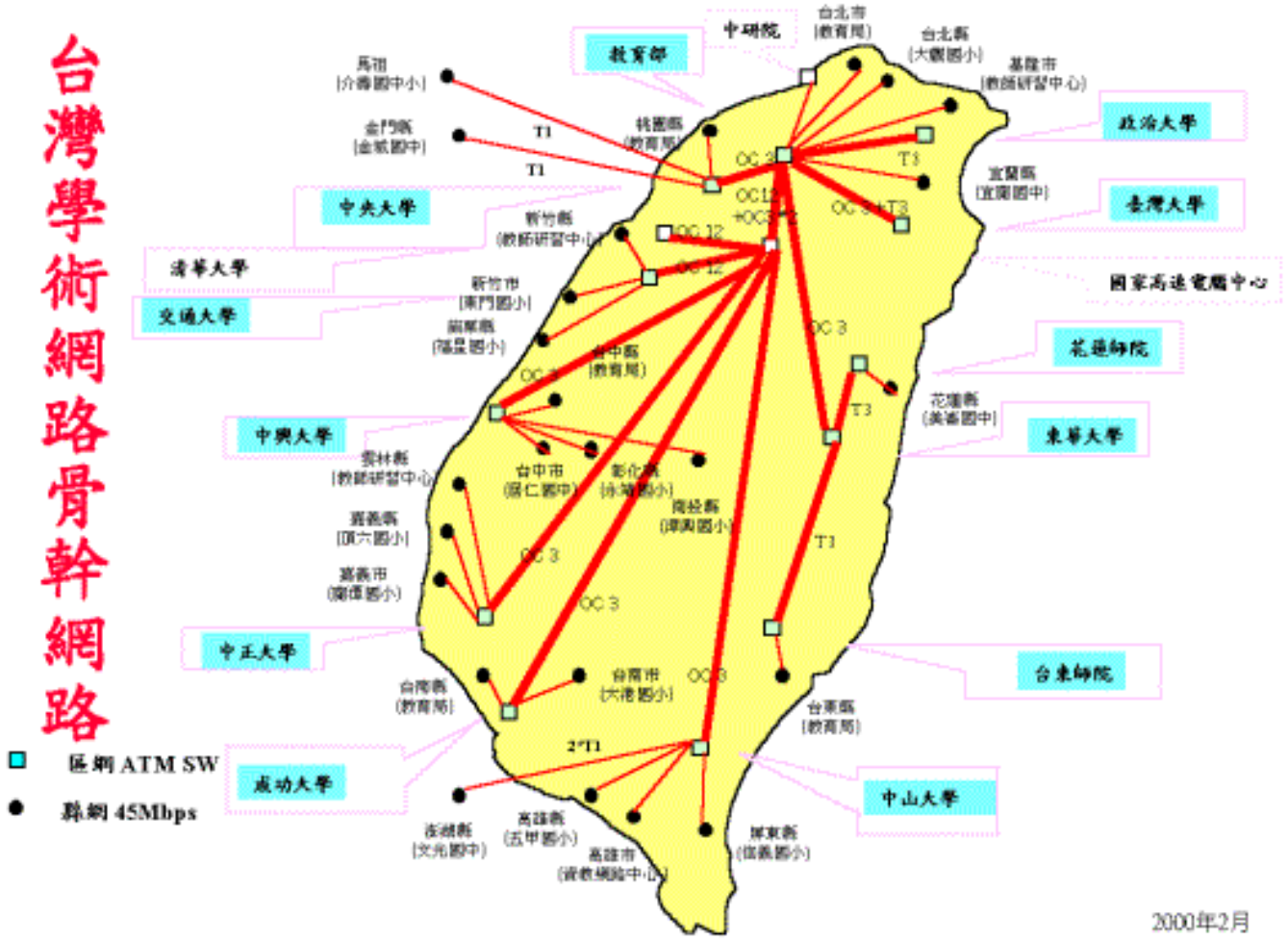
一些可以思考的東西

- time complexity = ?
- 答案:
- adjacency matrix: $O(|V|^3)$
- adjacency lists: $O(|V||E|)$

- 前面都沒有提到怎麼真的輸出path本身經過哪些vertex
- 想想看, 要怎麼做?

答案: 每次update distance的時候, 順便update附在旁邊的linked list. (用linked list表示path)

TANET


台灣學術網路骨幹網路
區網 ATM SW
縣網 45Mbps
教育部
中研院
台北市 (教育局)
台北縣 (大觀國小)
基隆市 (教師研習中心)
馬祖 (介壽國中小)
金門縣 (金城國中)
T1
T1
桃園縣 (教育局)
OC 3
T1
宜蘭縣 (宜蘭國中)
政治大學
臺灣大學
中央大學
新竹縣 (教師研習中心)
OC 12
OC 12
清華大學
交通大學
新竹市 (東門國小)
苗栗縣 (福星國小)
國家高速電腦中心
OC 3
花蓮師院
台中縣 (教育局)
OC 3
T3
花蓮縣 (美崙國中)
中興大學
東華大學
雲林縣 (教師研習中心)
台中市 (居仁國中)
彰化縣 (永靖國小)
南投縣 (埔里國小)
T1
嘉義縣 (頂六國小)
嘉義市 (蘭潭國小)
OC 3
OC 3
中正大學
台南市 (大港國小)
OC 3
台東師院
台南縣 (教育局)
台東縣 (教育局)
2*T1
成功大學
中山大學
澎湖縣 (文光國中)
高雄縣 (五甲國小)
高雄市 (資教網路中心)
屏東縣 (信義國小)
2000年2月

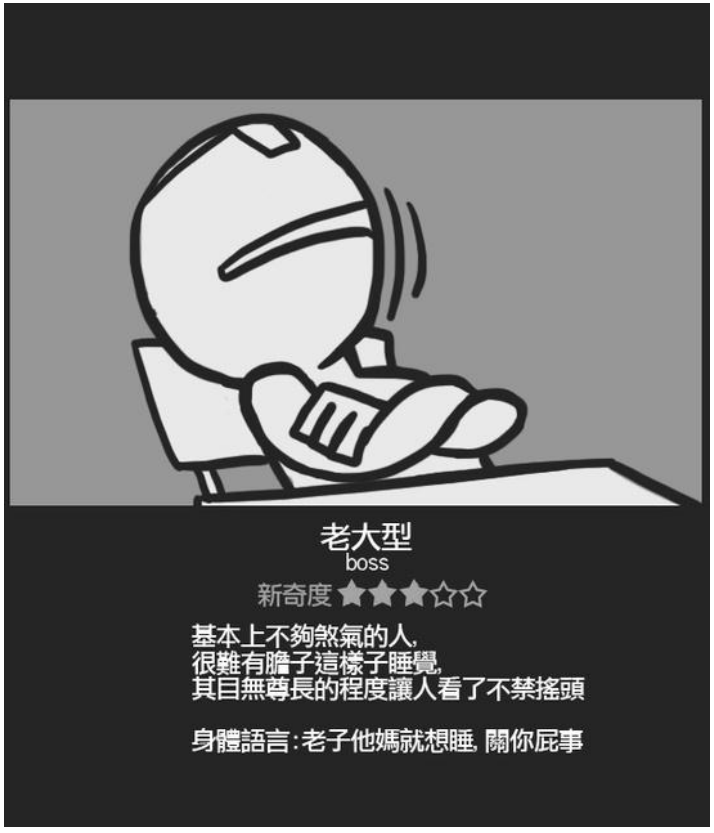
Routing protocol

- 要計算從某個點到另外一個點的路徑(route)
- 目標: 連線速度?
- path cost =?

- Dijkstra比較適合還是Bellman-ford比較適合?

答案: 似乎Bellman-Ford比較適合, 因為不需要知道所有link cost即可建立routing table

周末愉快


老大型
boss
新奇度
基本上不夠煞氣的人,
很難有膽子這樣子睡覺,
其目無尊長的程度讓人看了不禁搖頭
身體語言:老子他媽就想睡, 關你屁事