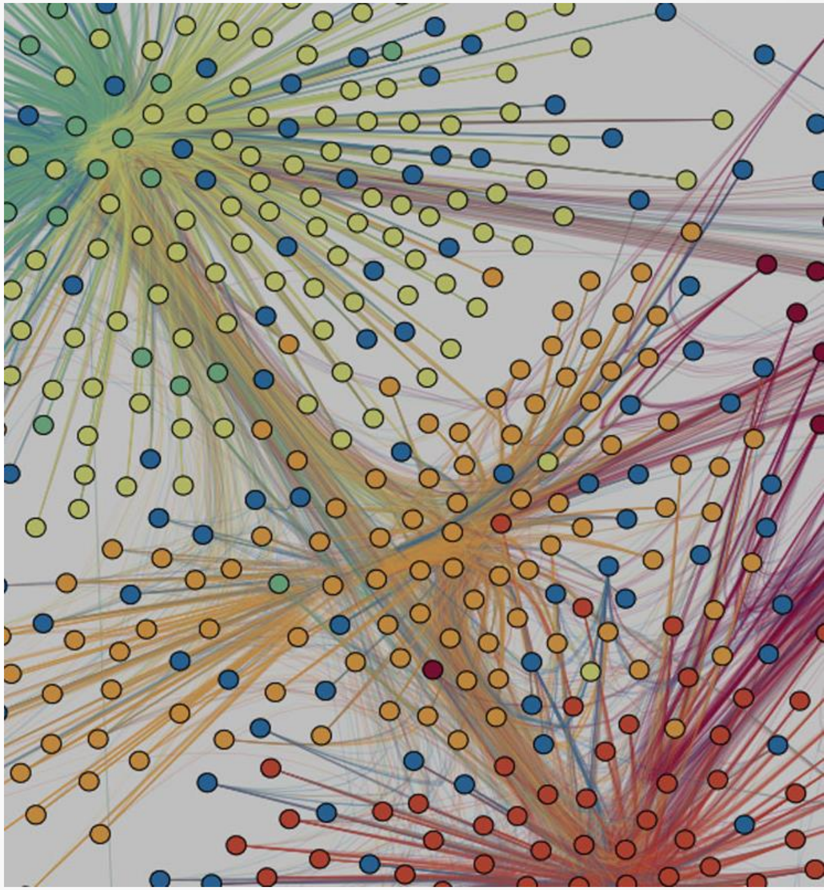
GRAPH AND SEARCH ALGORITHMS

許志仲(Chih-Chung Hsu)

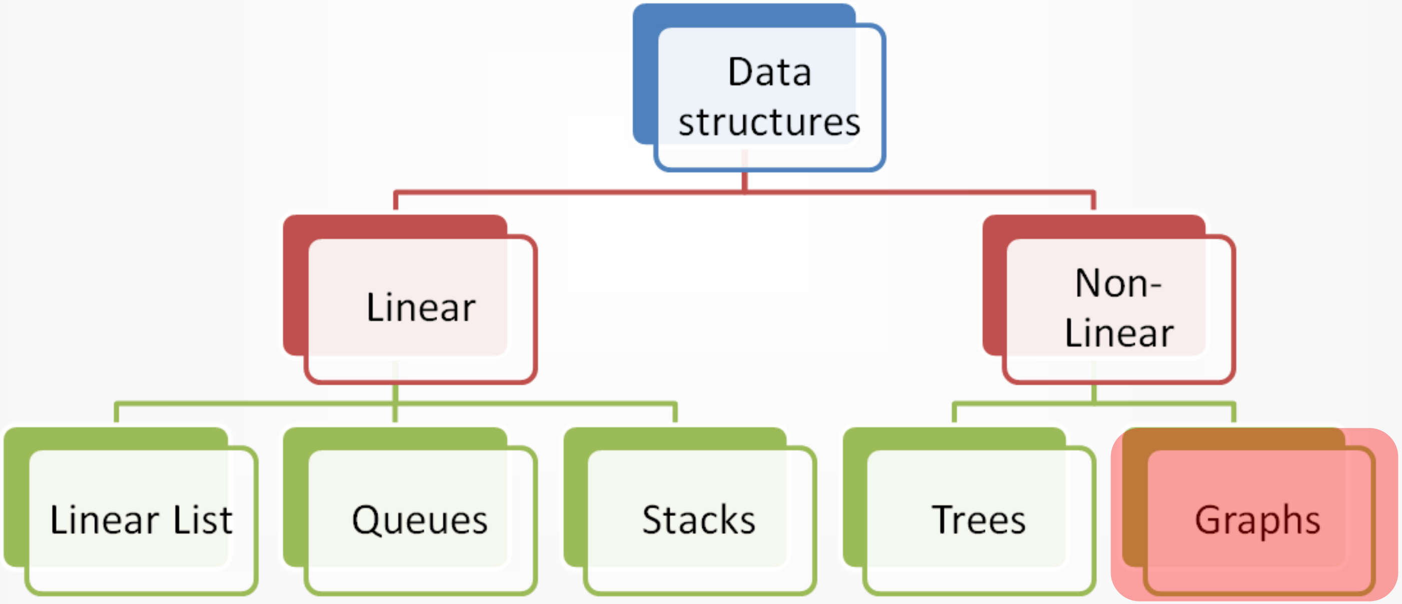
cchsu@gs.ncku.edu.tw



跨域智慧計算的世界

- 跨域智慧計算定義：以數據為基礎，結合多學科知識，透過人工智能與機器學習技術，解決複雜問題的創新過程。
  - 在全球數據急劇增長的今天，跨域智慧計算已成為創新與進步的驅動力。
- What we offer?
  - 融合了AI與編程的專業知識，更加入了跨學科團隊合作的F實作練習
- 學習如何挑選並應用適合的演算法，解決跨學科的實際問題。

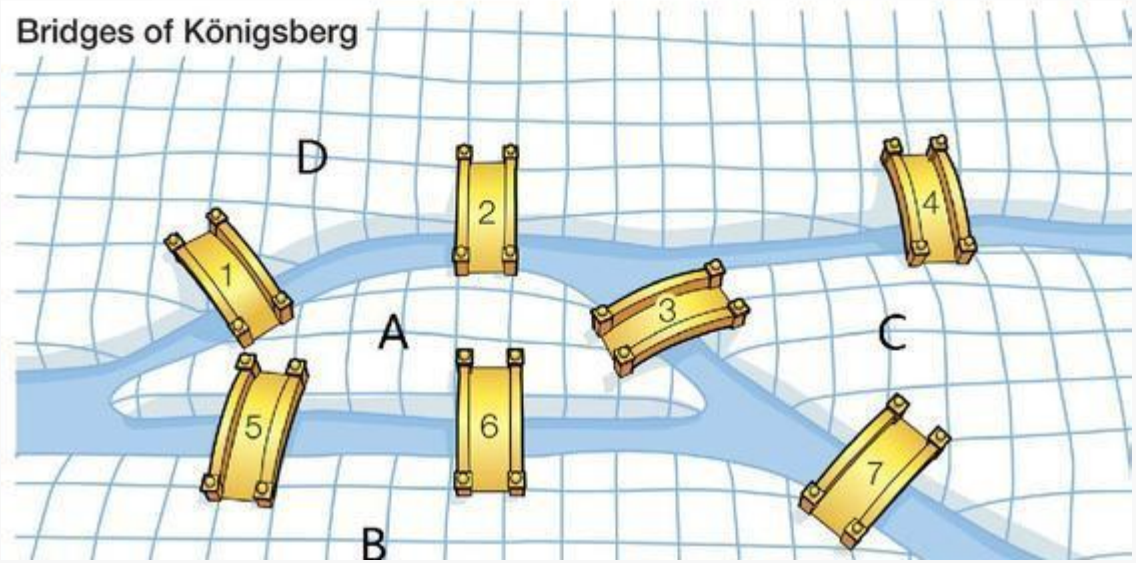
資料結構示意圖



同時也是最有(恐)用(怖)的一個章節!!

圖論

- Why?
  - 七橋問題。Eular (數學家) 為了求解肯尼茲堡橋問題
    - 是什麼? 隨意一地出發，一次經過所有的橋再回到原點，每座橋只走一次



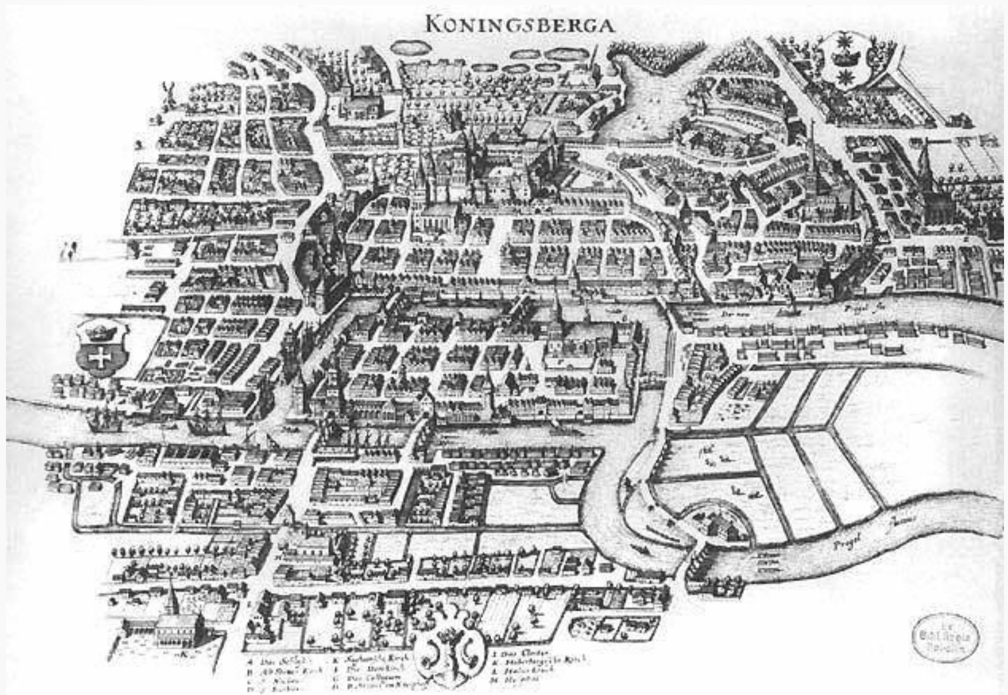
怎麼解決這個問題？

- 給定一張地圖：
  - 窮舉？
- 給定一千張地圖：
  - 逐一窮舉？
  - 找出規則？每張地圖都不一樣，怎麼去找規則？

- 要找規則（rule），必先將實際問題抽象化（abstraction）
  - 抽象化：將實際問題用模型（model）表示；通常是數學模型
  - 略去跟問題本質無關的細節，只留下關鍵元素
  - 抽象化是為了一般化（generalization）

抽象化

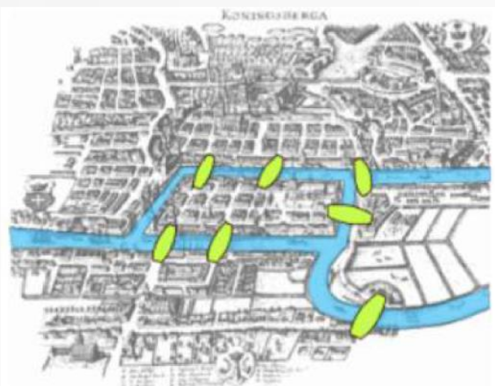
- 什麼是與問題本質無關的細節？



"Königsberg 1651" by Merian-Erbe is under Public Domain

抽象化

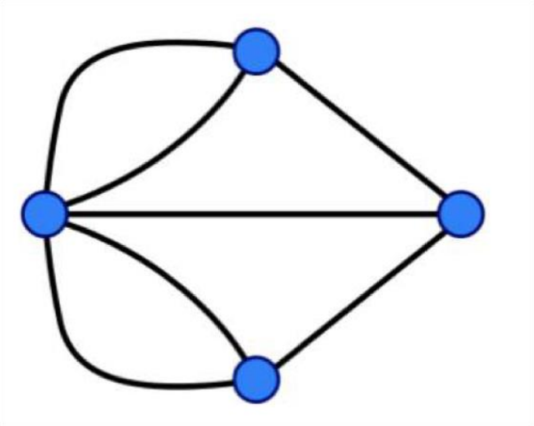
- 七橋問題的抽象化：



- 尤拉把實際的抽象問題簡化為平面上的點與線組合，每一座橋視為一條線，橋所連接的地區視為點
- 最終的模型是一個圖（graph）或網路（network）
  - 由點（vertex、node）和線（edge、link、arc）組成

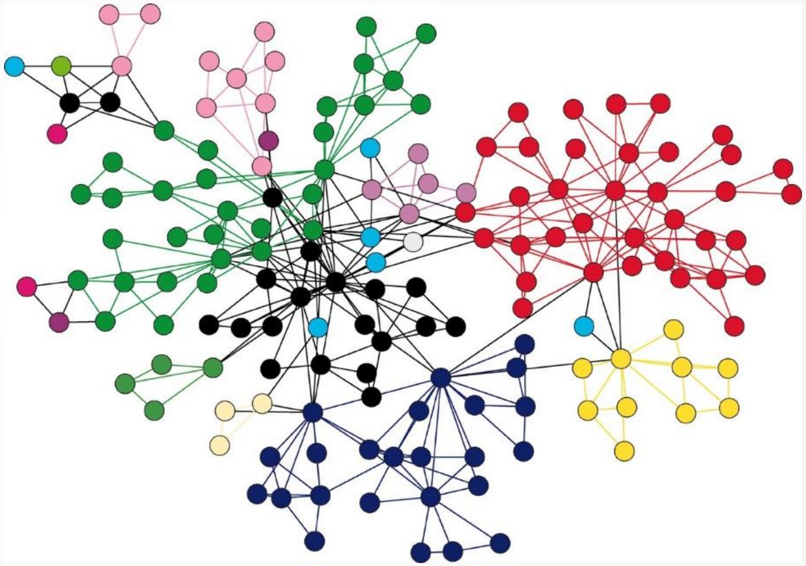
抽象化與一般規則

- 要解七橋問題，只要對著抽象化後的模型做探討即可
- 尤拉（Euler）在1735 年論述，這個圖不存一筆劃且不重複地走完所有的邊的解
  - 若從某點出發後最後再回到這點，則這一點上的連結數（稱為「degree」）必須是偶數，這樣的點稱為偶頂點。
  - 相對的，連有奇數條線的點稱為奇頂點。
  - 由於這個圖中存在四個奇頂點，所以必然無解



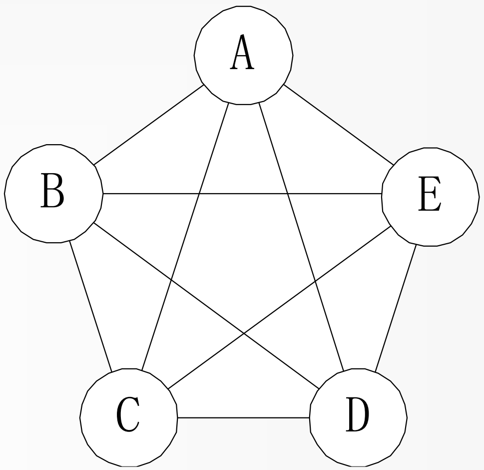
圖形結構能解的問題

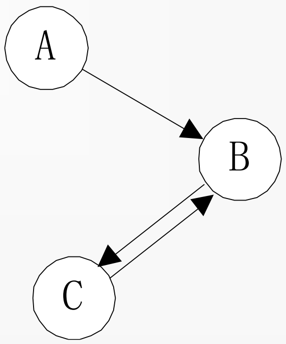
- 圖論很廣，通常是屬於離散數學的一種
- 以程式來講，能解決的問題通常是下列這些
  - 最短路徑
    - 路徑規劃
  - Spanning tree
    - 找到最小可以連通的路徑的技巧
  - 子圖搜尋
    - 找到最小的相似結構組合，例如在社群媒體中找到特定的小群



圖形結構定義

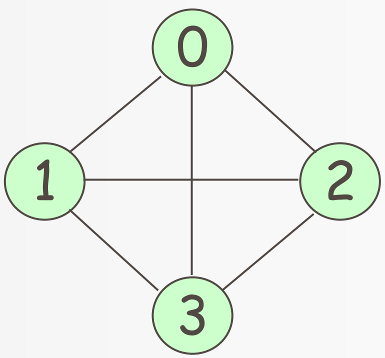
- 圖形(Graph)
  - G = (V，E)
- 頂點(Vertices，或稱Nodes)
- 邊(Edges)
- 無向圖(Undirected Graph)
  - (V1 ,V2)
- 有向圖(Directed Graph)
  - <V1 ,V2>



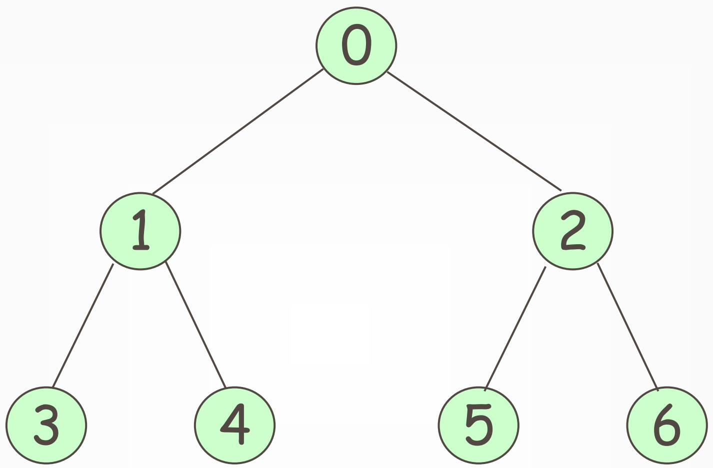


基本範例

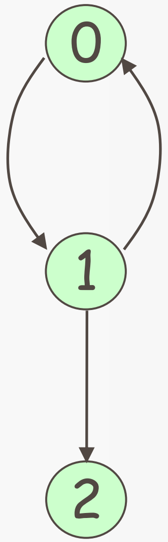
(a) G_1



(b) G_2



(c) G_3



$V(G_1) = \{0, 1, 2, 3\}$

$E(G_1) = \{(0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 3)\}$

$V(G_2) = \{0, 1, 2, 3, 4, 5, 6\}$

$E(G_2) = \{(0, 1), (0, 2), (1, 3), (1, 4), (2, 5), (2, 6)\}$

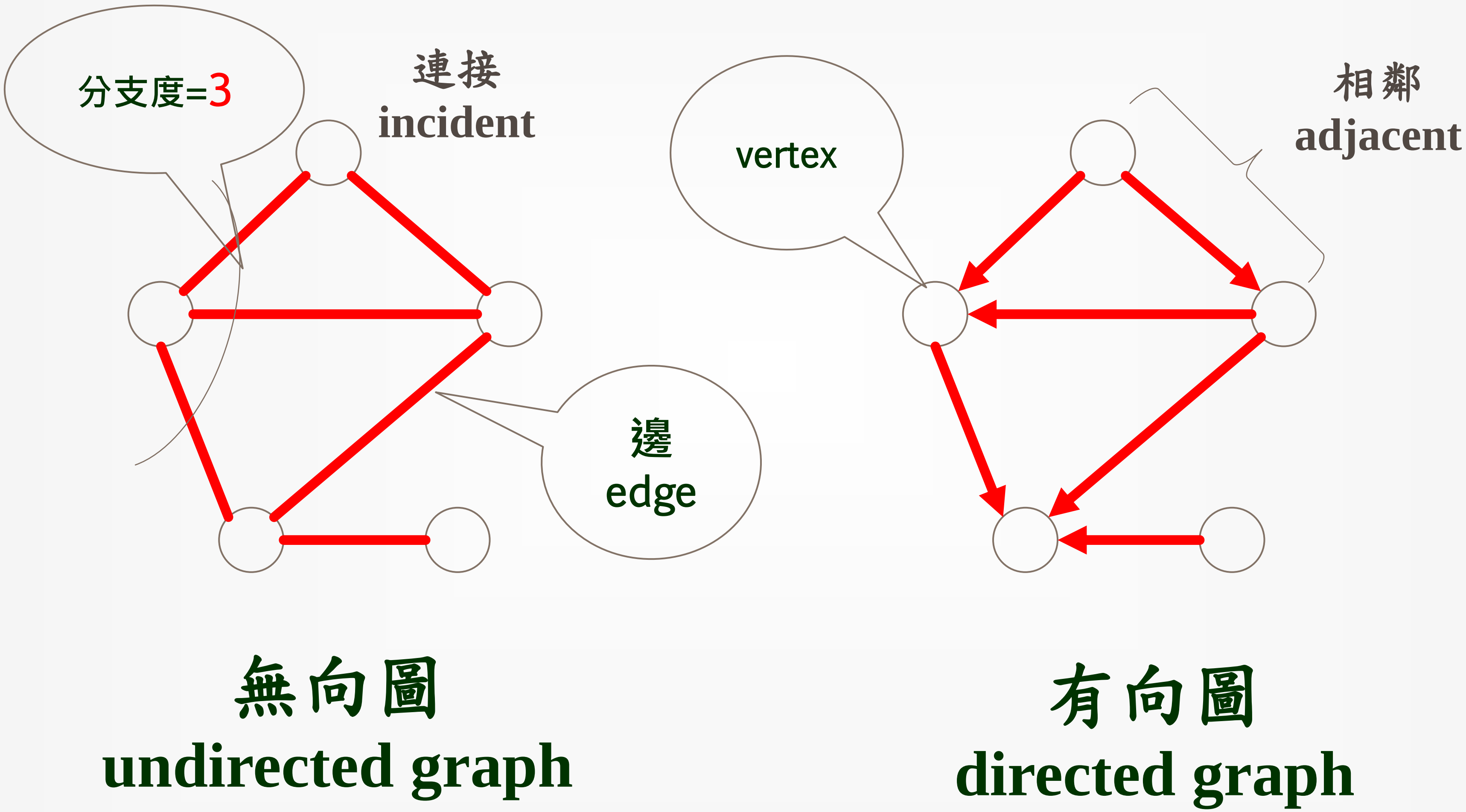
$V(G_3) = \{0, 1, 2\}$

$E(G_3) = \{<0, 1>, <1, 0>, <1, 2>\}$

Undirected Graph

Directed Graph

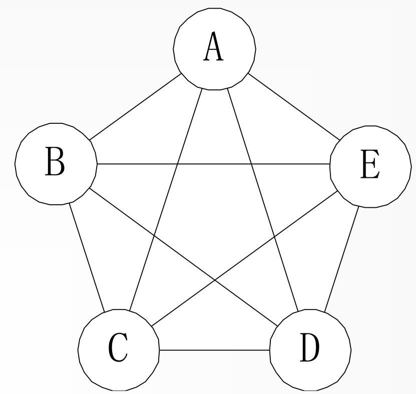
基本範例

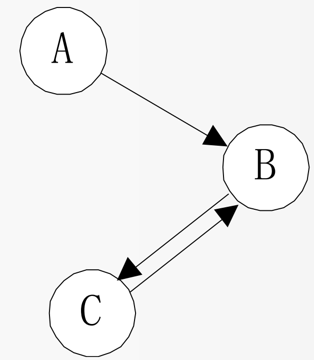


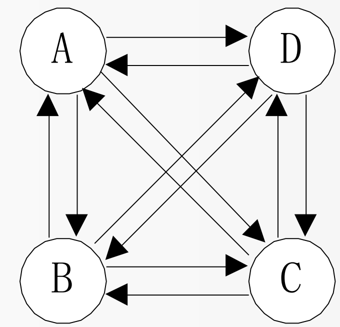
分支度=3
連接
incident
vertex
相鄰
adjacent
邊
edge
無向圖
undirected graph
有向圖
directed graph

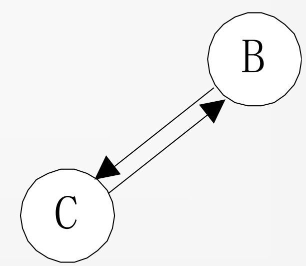
圖形的基本術語

- 完全圖(Complete Graph)
- 路徑(Path)
- 路徑之長度(Path Length)
- 簡單路徑(Simple Path)
- 迴路(Cycle)
- 相連的(Connected)
- 相連單元(Connected Component)
- 子圖(Subgraph)
- 緊密相連(Strongly Connected)
- 緊密相連單元(Strongly Connected Component)
- 出分支度(Out Degree)
- 入分支度(In Degree)



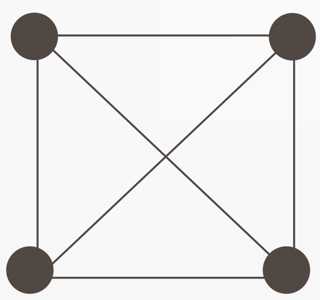




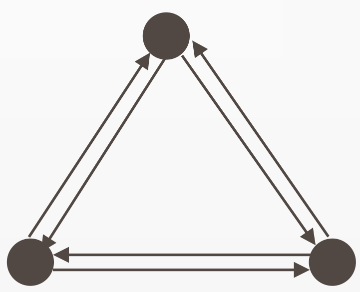


Complete Graph

- 整個圖每一個點，都有跟任意另一個點連線稱之
  - Vertex的數量可計算
- 依據圖形分類
  - Undirected graph
    - an n-vertex graph with exactly n(n-1)/2 edges.
  - Directed graph
    - an n-vertex graph with exactly n(n-1) edges



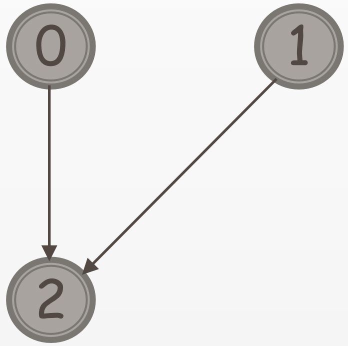
Undirected



Directed

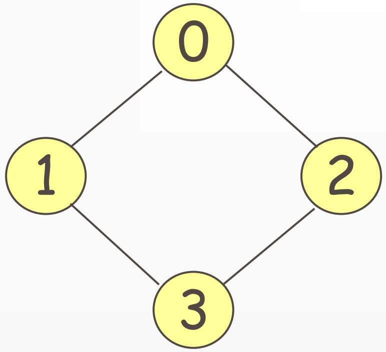
路徑與其長度(Path and Length)

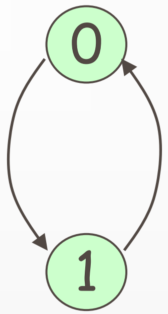
- 路徑(Path)
  - 定義：在圖形G中，從Vertex v到u所經過的edge，就叫做路徑。例如：u, i1, i2, …, ik, v ➔ (u, i1), (i1, i2), …, (ik, v).
- 路徑長度(Length)
  - 簡單把上面的路徑數量統計一下就是了…
- Simple Path
  - 在一路徑中，除了起點與終點可以相同之外(不同亦可)，其餘頂點不可以重複。



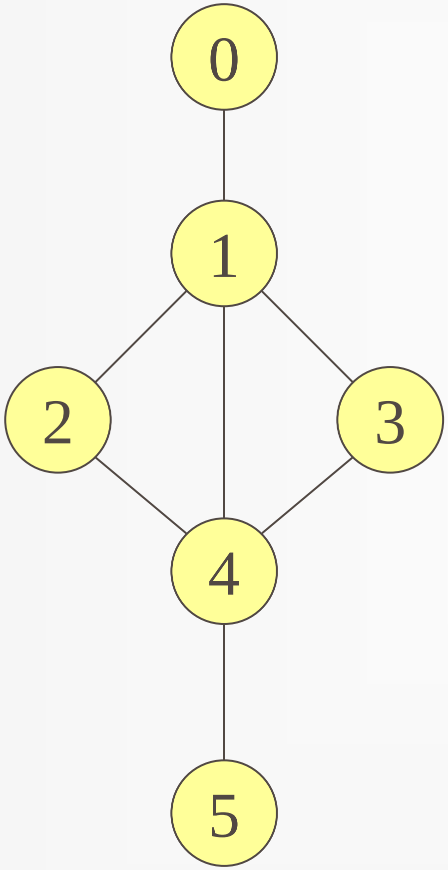
循環(Cycle)

- 圖形中的迴路是一條路徑，且必須符合下列兩要求
  - 是一條簡單路徑(Simple path)
  - 起點與終點是同個頂點。





表示一個路徑的基本範例



Path 1 ***(0,1) (1,3) (3,4) (4,2) (2,1) (1,4) (4,5)***

Path 2 ***(0,1) (1,3) (3,4) (4,5)***

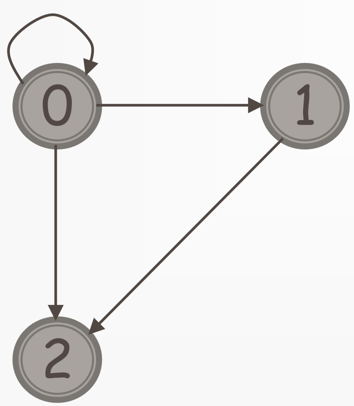
Path 3 ***(1,3) (3,4) (4,2) (2,1)***

Simple Path ***Path 2、Path 3***

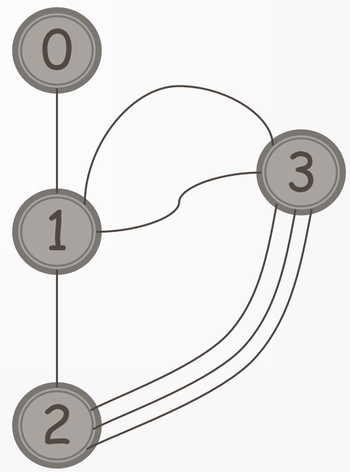
Circle ***Path 3***

Simple Graph

- 沒有自我迴圈
  - 不能有一個Edge 是自己連到自己!!
- 沒有多重邊或平行邊
  - 不能兩個Vertices 之間有多個路徑!!
- 簡單路徑構成的圖，就是Simple graph



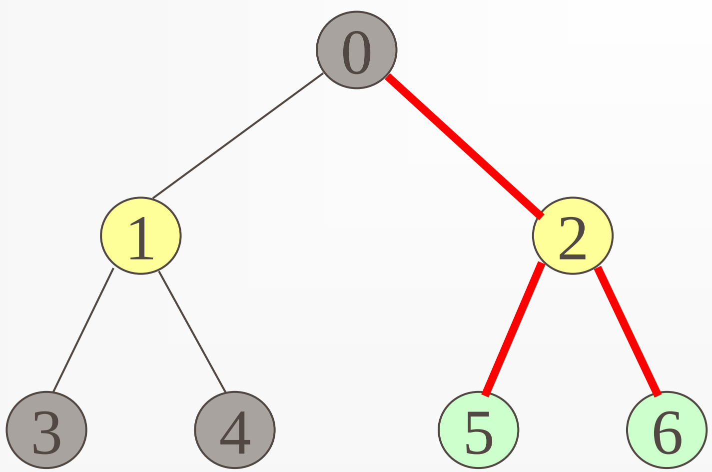
(a) Graph with a self edge

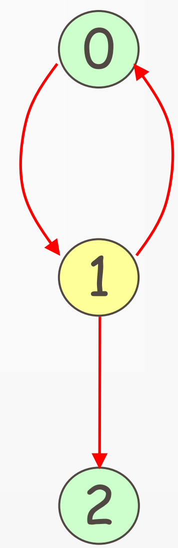


(b) Multigraph

連接與相鄰定義(Adjacent and Incident)

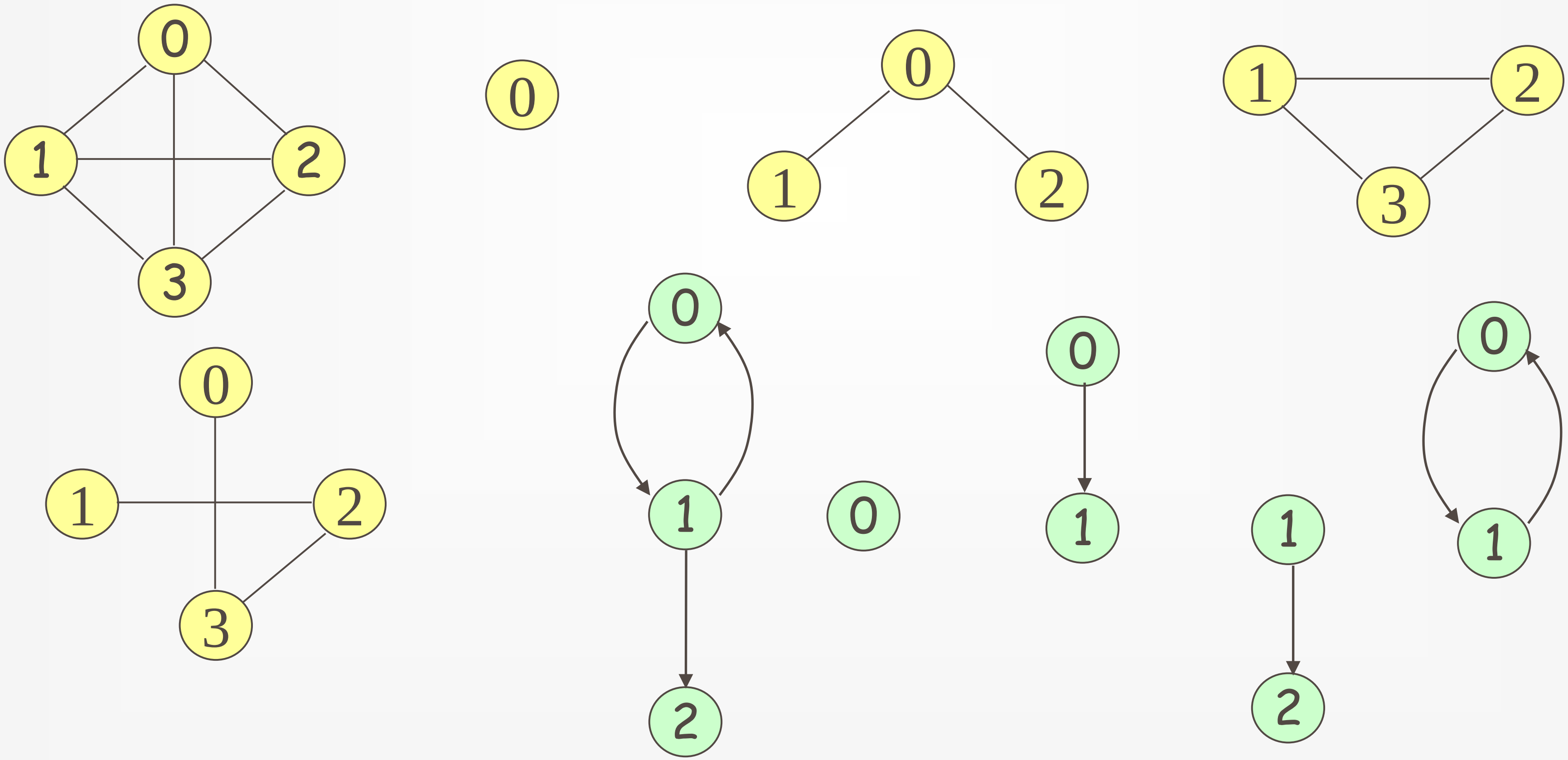
- 不同圖形類別有不同的定義
  - Undirected graph
    - If (u, v) is an edge in E(G), vertices u and v are adjacent and the edge (u, v) is the incident on vertices u and v.
  - Directed graph
    - <u, v> indicates u is adjacent to v and v is adjacent from u.
    - <u, v> is incident to u and v.





子圖(Subgraph)

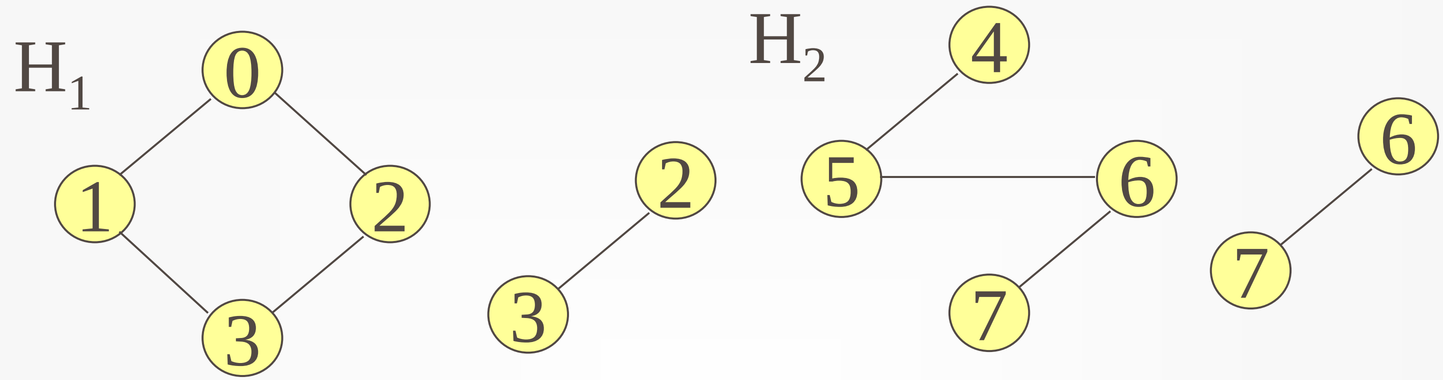
- 一圖形為G，其子圖為 G′ ，則 V(G′)$\subseteq$ V(G) 、 E(G′)$\subseteq$ E(G).

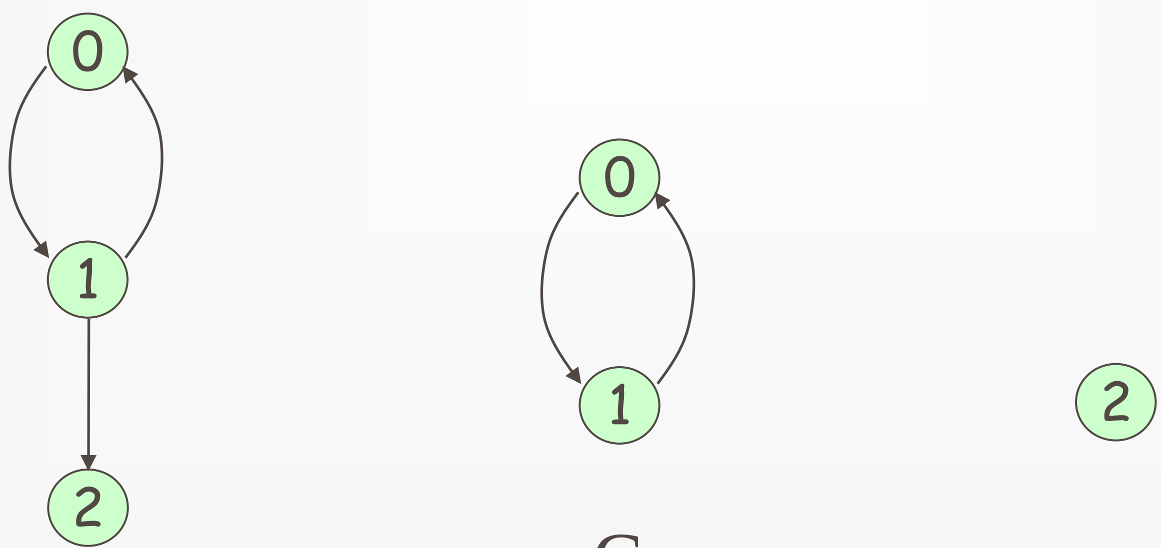


連通圖Connected

- 若圖G，任2點間有一路徑存在，該圖稱為連通圖。
- 若圖G不連通，則圖G的最大連通子圖，稱為圖G 的連通成分(connected components)
- 若圖G為有向圖，若且惟若圖G中任兩點u到v有一路徑存在，則v到u亦存在有一路徑，則圖G 稱為強連通(strongly connected)
- 有向圖G中，最大強連通子圖，稱為圖G 的強連通成分(strongly connected components)

連通圖的基本範例



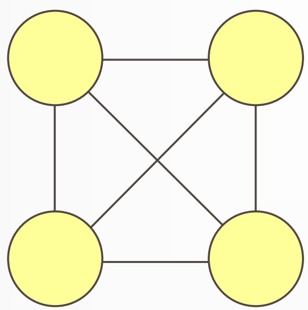


G_3

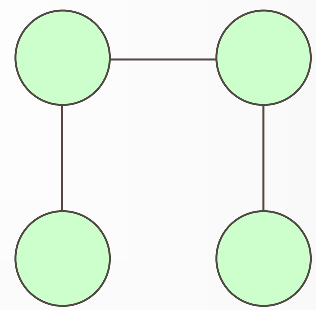
森林和樹(Forest & Tree)

- 森林是沒有迴路(Cycle) 的圖。
- 樹是連通的森林。
- 生成樹(Spanning tree) 是圖G 的形成樹的生成子圖。

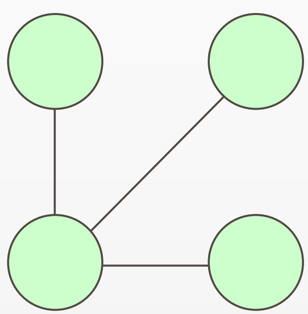
完全圖



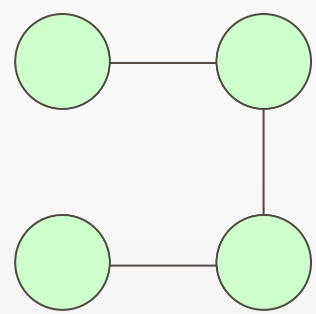
SP-1



SP-2

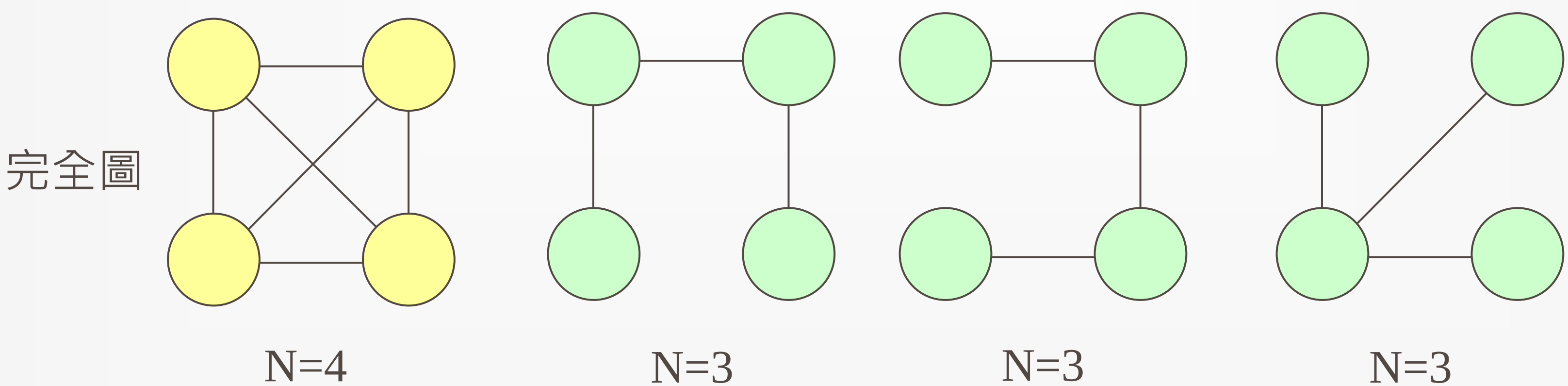


SP-3



生成樹的特質

- 一個包含N 個頂點的無向相連圖，我們可以找出用圖中的N-1 個邊來連接所有頂點的樹
- 若再加入圖形中其餘的邊到生成樹中必會形成迴路
- 生成樹中的任兩個頂點間都是相連的，也就是存在一條路徑可通，但此一路徑不一定是原圖形中該兩頂點之最短路徑。



Graph 的一些性質

- 計算Degree 的方法
  - 令G為一具有m邊的圖，則

$$\sum_{v\in G}\deg(v)=2m$$

- 內連/外連Degree 的關係!!
  - 令G為一具有m邊的有向圖，則

$$\sum_{v\in G} in\deg(v)=\sum_{v\in G} out\deg(v)=m$$

Graph 的一些性質

- 量化Edge 數量與Vertex 數量之間的關係
  - 令G為一具有n 個頂點和m 個邊的簡單圖。
    - 若G為無向圖，則

$$m \le n(n-1)/2$$

    - 若G 為有向圖，則

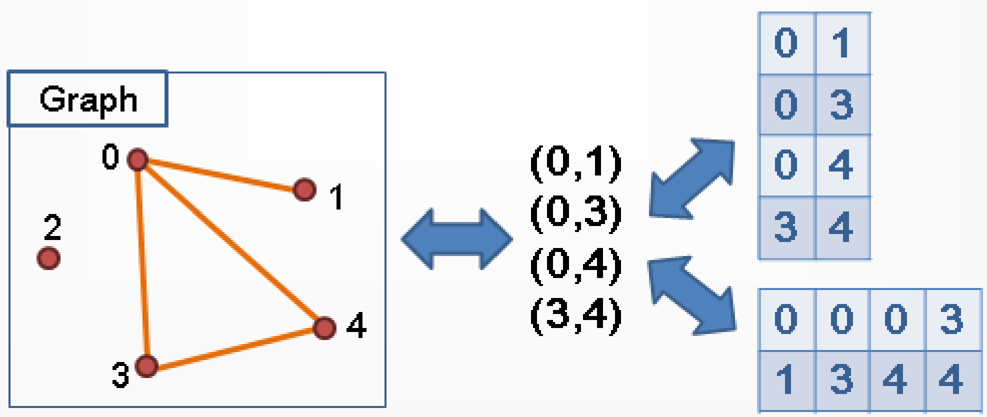
$$m \le n(n-1)$$

Graph 的一些性質

- 確認Graph, tree, forest 之間的關係
- 圖G 為一有n 個頂點與m 個邊的無向圖，則
  - 若G 為連通圖，則m >= n – 1
  - 若G 為樹，則m = n – 1
  - 若G 為森林，則m <= n - 1

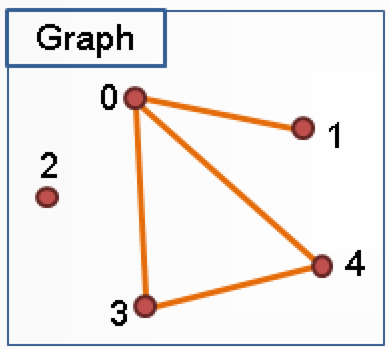
圖形表示法

- 邊的清單(Edge list)
  - 兩兩一對的當成儲存空間
  - 最省空間!!
    - 不利於計算

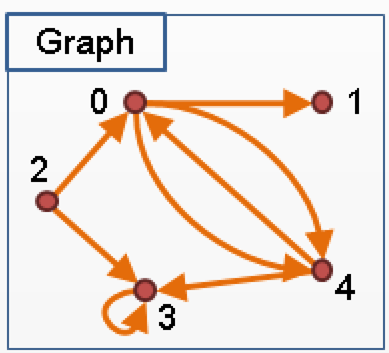


圖形的表示法

- 對於圖形結構，要怎麼儲存?
  - 鄰接矩陣(Adjacency matrix) 表示法是一種



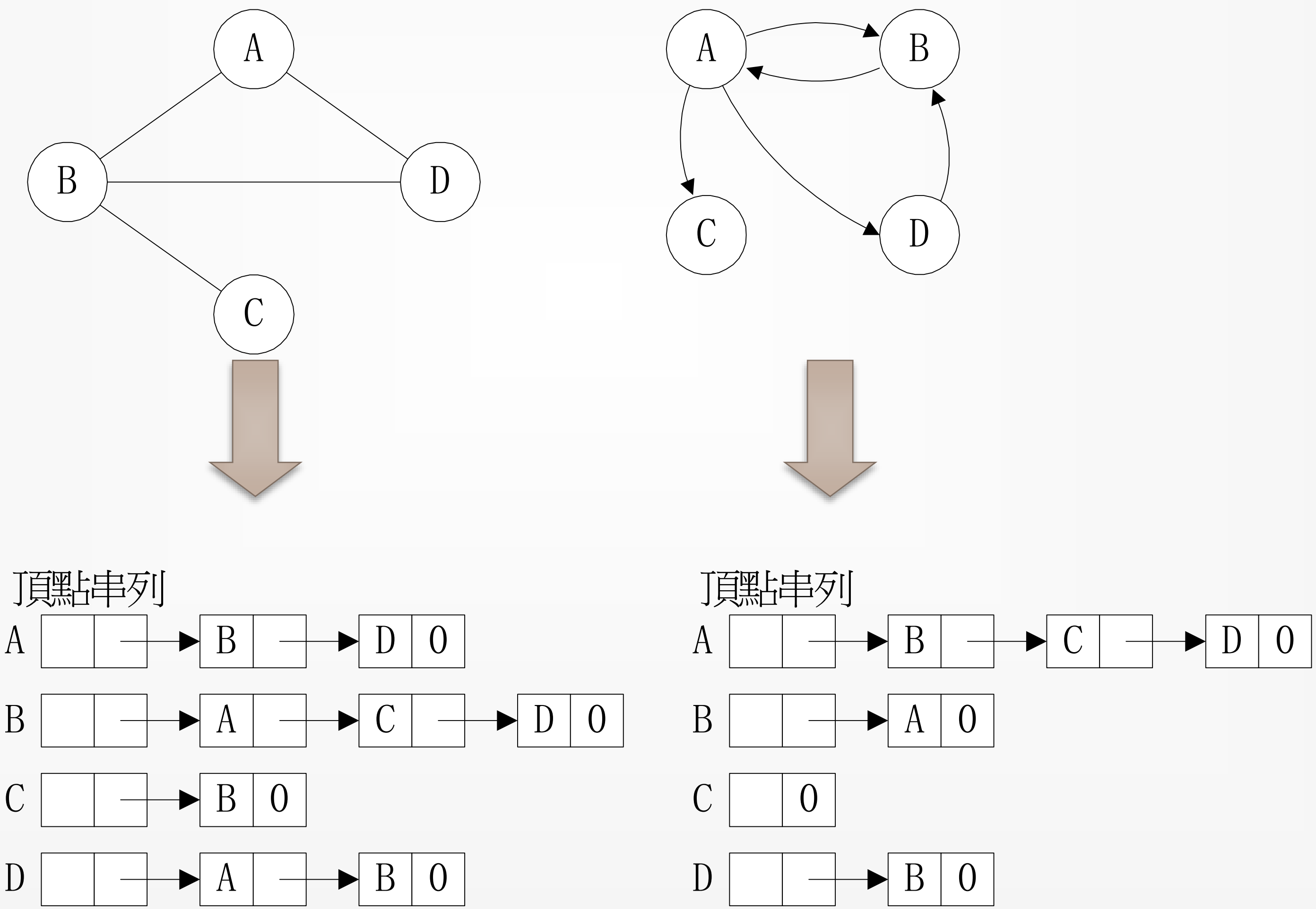
	0	1	2	3	4
0	0	1	0	1	1
1	1	0	0	0	0
2	0	0	0	0	0
3	1	0	0	0	1
4	1	0	0	1	0



	0	1	2	3	4
0	0	1	0	0	2
1	0	0	0	0	0
2	1	0	0	1	0
3	0	0	0	1	0
4	1	0	0	1	0

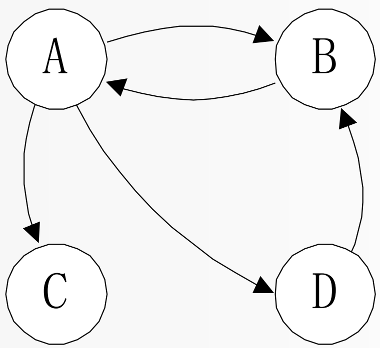
圖形的表示法

- 鄰接串列(Adjacency list) 表示法

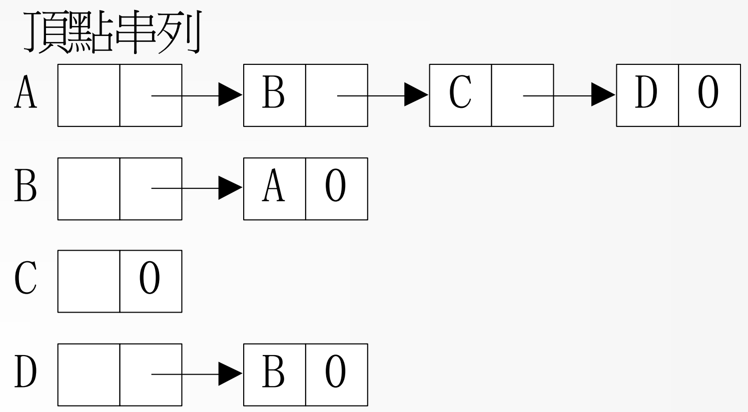


圖形的表示法

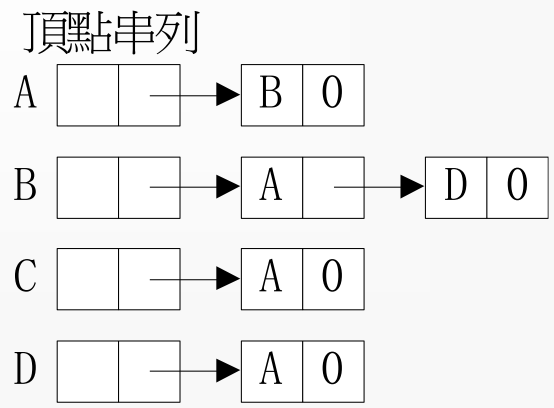
- 鄰接串列與反鄰接串列



連接串列



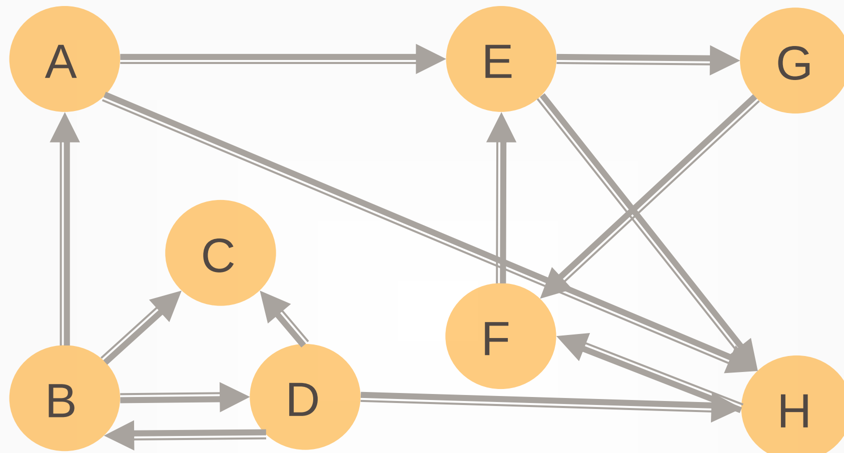
反連接串列



看的箭頭方向相反!!

圖形表示法

- 複習一下兩種表示法-- 各有優缺點



相鄰矩陣表示法

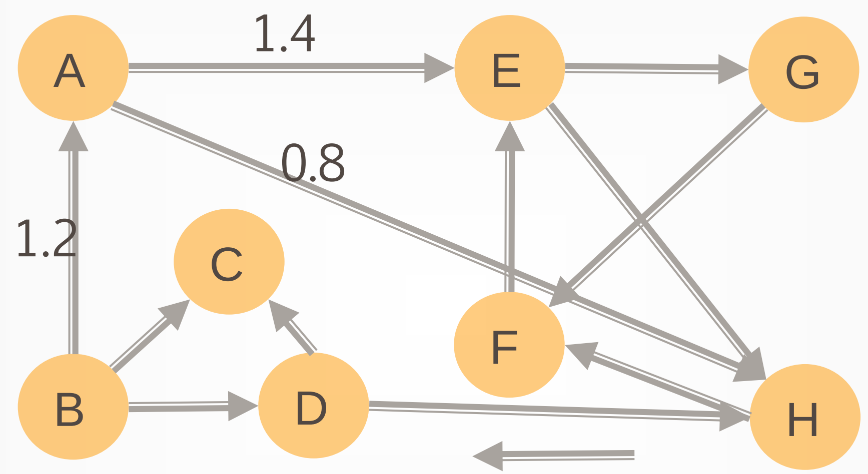
	A	B	C	D	E	F	G	H
A	0	0	0	0	1	0	0	1
B	1	0	1	1	0	0	0	0
C	0	0	0	0	0	0	0	0
D	0	1	1	0	0	0	0	1
E	0	0	0	0	0	0	1	1
F	0	0	0	0	1	0	0	0
G	0	0	0	0	0	1	0	0
H	0	0	0	0	0	1	0	0

邊的清單表示法

A	E	H	-
B	A	C	D
C	-	-	-
D	B	C	H
E	G	H	-
F	E	-	-
G	F	-	-
H	F	-	-

相鄰圖形表示法加強板

- 萬一Edge表示Vertices之間的關係呢?

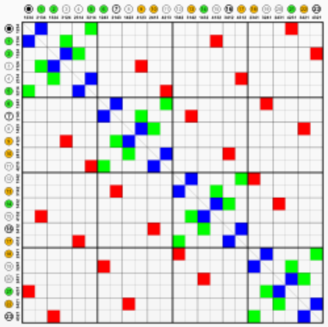


相鄰矩陣
表示法

	A	B	C	D	E	F	G	H
A	0	0	0	0	**1.4**	0	0	**0.8**
B	**1.2**	0	1	1	0	0	0	0
C	0	0	0	0	0	0	0	0
D	0	1	1	0	0	0	0	1
E	0	0	0	0	0	0	1	1
F	0	0	0	0	1	0	0	0
G	0	0	0	0	0	1	0	0
H	0	0	0	0	0	1	0	0

圖形結構的精髓

- 一般圖形結構僅僅只要利用相鄰矩陣就可以儲存
  - 跟一般陣列沒什麼兩樣?!!
    - 沒錯!!
  - 重點在於這樣的結構，有很多漂亮的演算法可以用
    - 搜尋路徑(Graph traversal)
    - 最短路徑搜尋(Shortest path)
    - 找關係最相像的資料(Relationship inferring)
    - 把資料分成幾群(Clustering)
    - 把要找的某個結構從圖中找出來(Sub-graph problem)
    - 太多了…



Example

- https://tinyurl.com/ywztd2p3

- 想像一個大型魚塭養殖場,裡面有數十個相連的魚塭池,每個池子都裝有感測器監控水質和魚苗狀況。如果發現某個池子的數據異常,就需要通知工作人員前往檢查並採取必要的行動。為了有效地監控整個系統,我們可以使用DFS演算法來定期遍歷每個池子的感測數據。

DFS

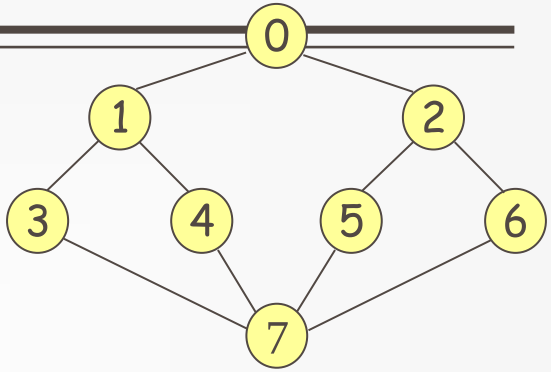
- **適用情境**:
  - 需要遍歷整個網路/圖形中的所有節點
  - 檢測網路/圖形中是否存在特定的路徑或循環
  - 解決像是迷宮、家譜等具有階層/樹狀結構的問題
  - 對記憶體使用量的要求較低
- **不適用情境**:
  - 尋找最短路徑問題 (BFS更好)
  - 處理極大規模的網路/圖形 (可能發生堆疊溢出)
  - 存在大量平行分支的情況 (DFS效率較低)
  - 需要按照特定順序訪問節點
  - DFS適合遍歷具有層級/樹狀結構的網路/圖形,但在面對龐大或高度平行化的問題時,可能需要其他更合適的演算法,避免效率低落或資源耗盡的情況發生。選擇使用哪種演算法,需要根據具體問題的特點來權衡。

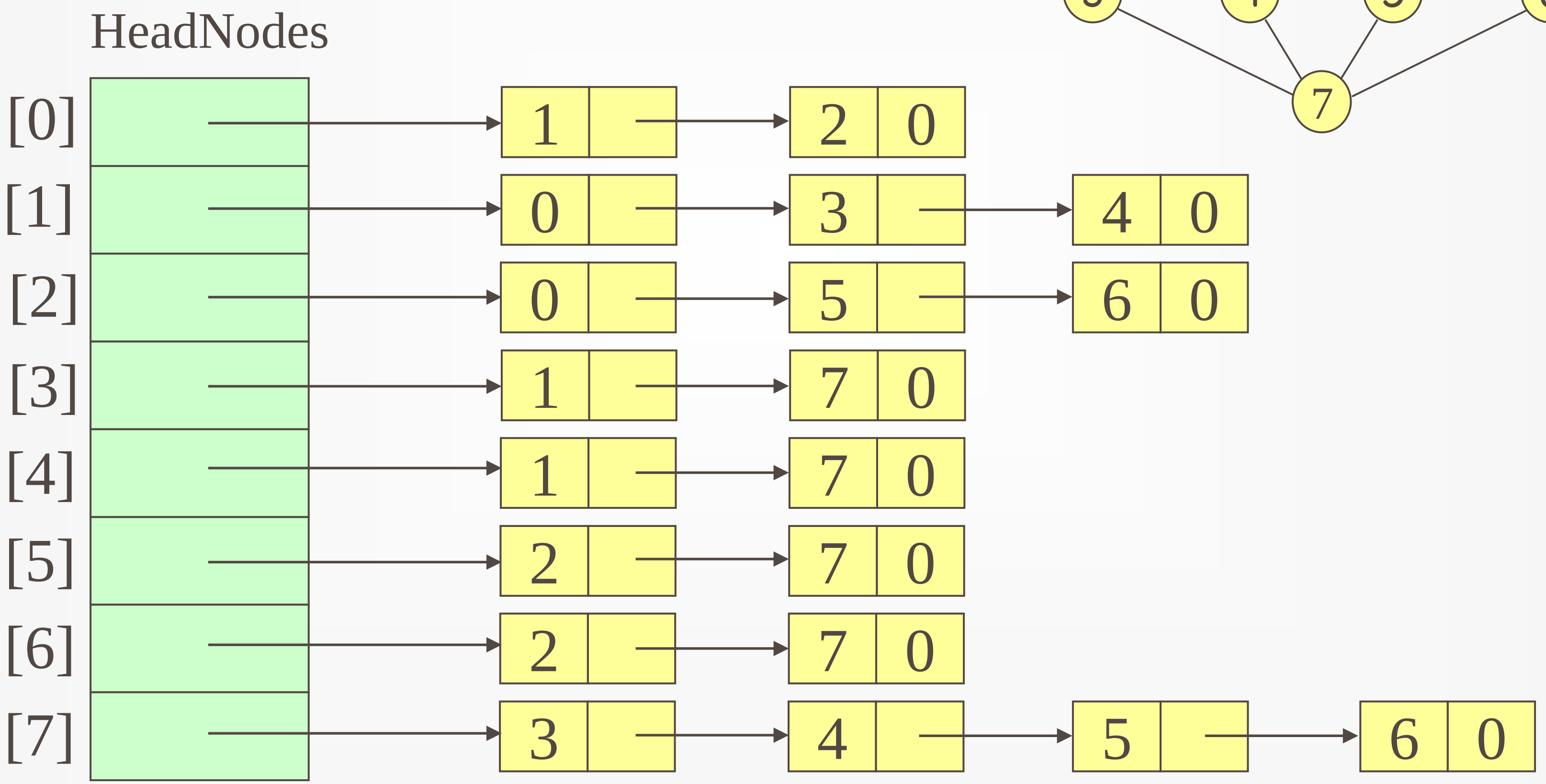
Graph Traversal 問題

- 要有效率地把所有的Vertices都找過一遍不容易
  - 樹狀結構，很有規則，可有前中後序的找法
  - Graph?
- 普遍來說，共有兩種解法
  - Breadth-first search (BFS)
  - Depth-first search (DFS)

深度優先(Depth-First Search)

0,1,3,7,4,5,2,6





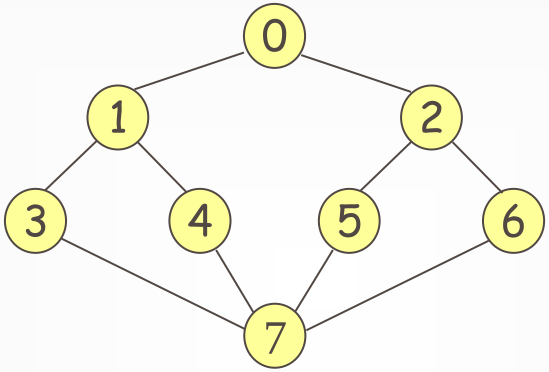
深度優先搜尋

- 由樹的根(或圖的某一點當成根) 開始
- 先到相鄰的(有一edge的意思) Vertex上，並且未搜尋過的節點上
- 所謂深度；即讓此節點所有附近的點都跑過了，再回去上一層…
- 有沒有很熟悉的感覺?!
  - 遞迴+ Stack / Queue…!!!

深度優先分解動作

0,1,3,7,4,5,2,6

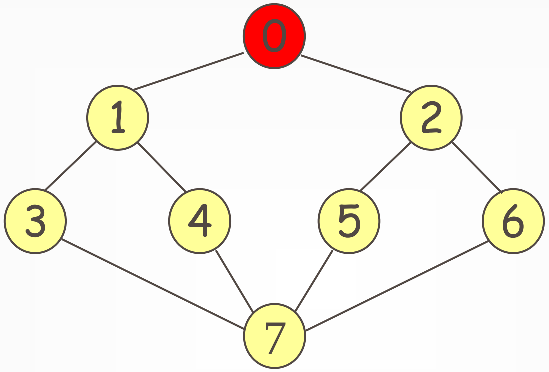
- 隨機找一個當root，先找0



深度優先分解動作

0,1,3,7,4,5,2,6

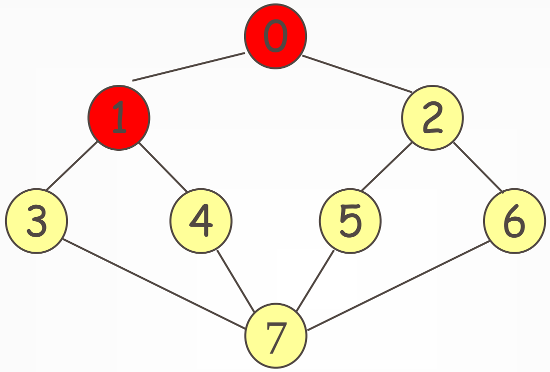
- 找相鄰的沒找過的node當下一個node! 就是1



深度優先分解動作

0,1,3,7,4,5,2,6

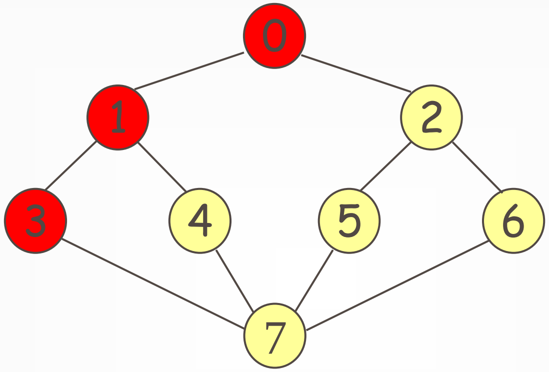
- (目前在1): 找相鄰的沒找過的node當下一個node! 就是3



深度優先分解動作

0,1,3,7,4,5,2,6

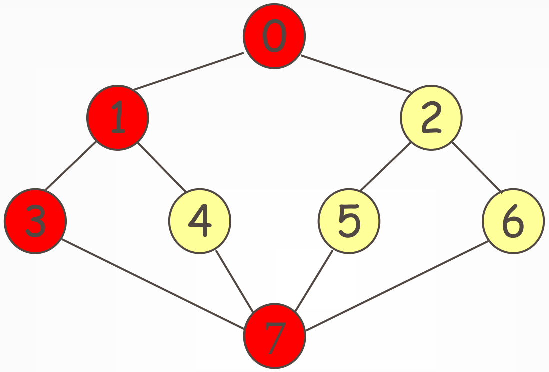
- (目前在3): 找相鄰的沒找過的node當下一個node! 就是7



深度優先分解動作

0,1,3,7,4,5,2,6

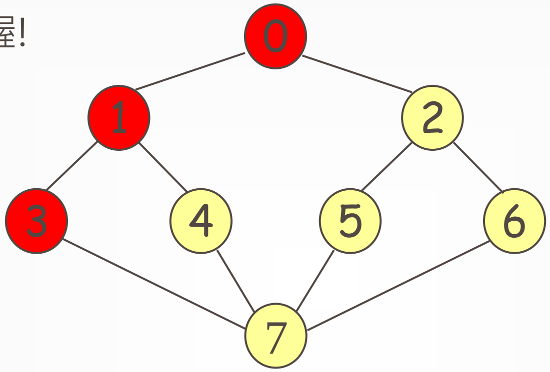
- (目前在7): 找相鄰的沒找過的node當下一個node! 就是4



深度優先分解動作

0,1,3,7,4,5,2,6

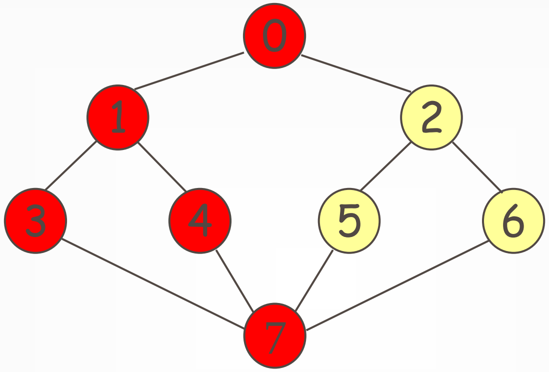
- (目前在4): 找相鄰的沒找過的node當下一個node!
  - 沒有這回事!! 都看過了喔!
  - 回上一層，所以回到7



深度優先分解動作

0,1,3,7,4,5,2,6

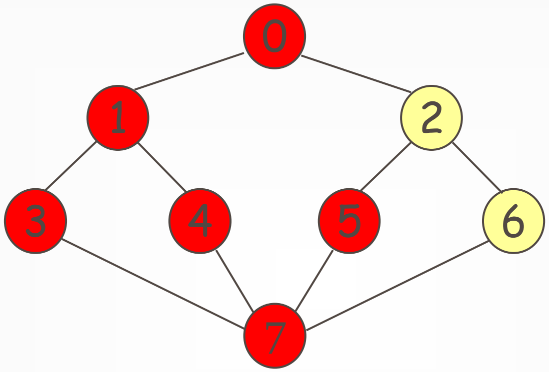
- (目前在7): 找相鄰的沒找過的node當下一個node! 就是5
  - 4已經走訪過了



深度優先分解動作

0,1,3,7,4,5,2,6

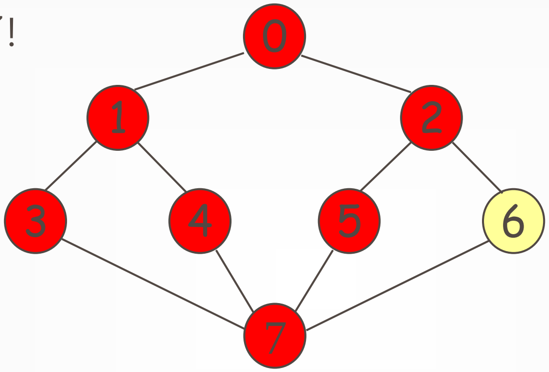
- (目前在5): 找相鄰的沒找過的node當下一個node! 就是2



深度優先分解動作

0,1,3,7,4,5,2,6

- (目前在2): 找相鄰的沒找過的node當下一個node! 就是6
  - 因為0一開始就被找過了!



DFS適用情境

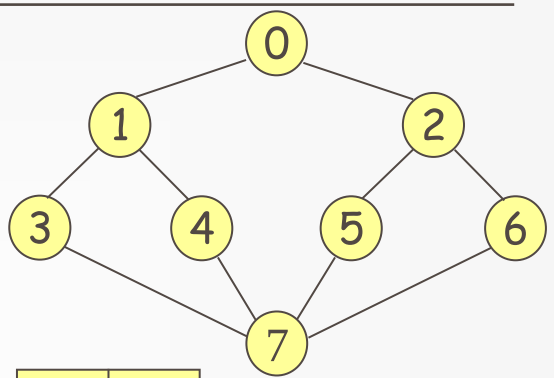
https://tinyurl.com/ywztd2p3

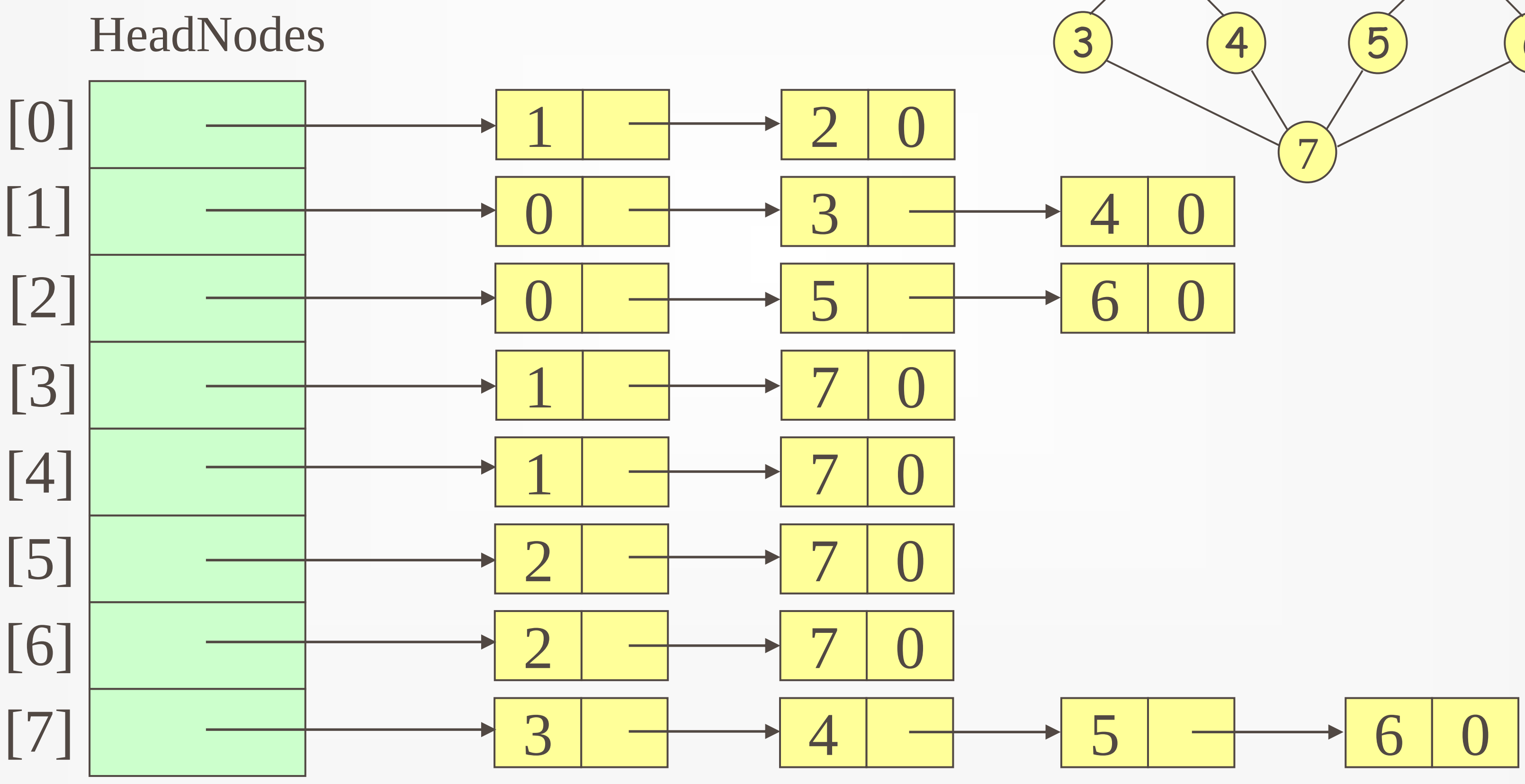
- https://tinyurl.com/ywztd2p3

- 這個Graph中，每個用戶都可以關注其他用戶，而這些關注關係則通過鄰接表來表示。
- **分析應用**
- 在這種社群媒體的圖表表示中，我們可以進行多種分析，例如：
- **影響力用戶的識別**：通過分析用戶被關注的數量，我們可以識別出哪些用戶在社群網絡中具有較高的影響力。
- **社群發現**：使用圖的遍歷算法，如BFS或DFS，我們可以探索社群網絡中的緊密連接群體或社群。
- **推薦系統**：根據用戶的關注關係和互動，我們可以建立推薦系統，為用戶推薦可能感興趣的新朋友或內容。

廣度優先(Breadth-First Search)

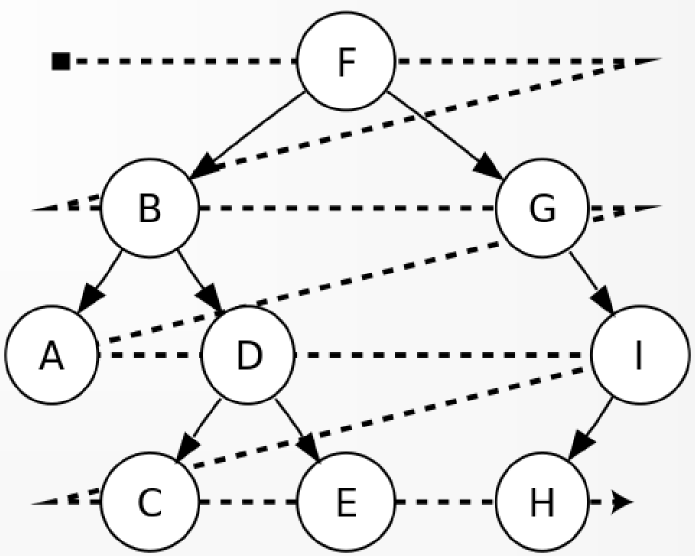
0,1,2,3,4,5,6,7





BFS 的方法

- 建立一個具有Vertices個數的Queue
- 照順序讀取資料進來，並根據下列規則處理
- 不斷找出尚未遍歷的點當作起點：
  - 把起點放入Queue。
  - While (queue != empty)
    - 從Queue 當中取出一點。
    - 找出跟此點相鄰且還沒看過的點，照順序存入Queue。



BFS 的方法(Pseudo code)

- 1. 首先準備一個佇列為Qu
- 2. 再將起始頂點v 加入到Qu 之中（亦即進行Enqueue動作）
- 3. 如果Qu 不為空，則執行下列步驟，否則跳到步驟4:
  - 3.1 從Qu 中取出一頂點w (亦即進行Dequeue動作）
  - 3.2 並將w 標示為『已拜訪過』
  - 3.3 將所有與w 相鄰且尚未標示『已拜訪過』的頂點加入到Qu 之中（亦即進行Enqueue動作）
  - 3.4 回到步驟3
- 4. 結束

思考一下

https://tinyurl.com/ywztd2p3

- 現在Graph結構都沒有weight，weight的形式？

- **BFS適用情境:**
  - 尋找最短路徑(例如兩個人之間的最短介紹路徑)
  - 網路廣播路由
  - 網路爬蟲爬取最臨近的網頁
  - 解決迷宮問題
- **BFS不適用情境:**
  - 圖太大時耗費過多記憶體
  - 需要優先處理某些特定節點時(可用Dijkstra等其他算法)
  - 深度遷移代價更高的問題(可用DFS等算法)
  - BFS擅長解決最短路徑問題、均勻遍歷所有節點。但在面對大規模問題或特殊優先級需求時,可能需要其他更合適的圖算法。

BFS不適用情境

- **社交媒體網路**
  - 像是Facebook、Instagram、Twitter等龐大的社交媒體網路,擁有數十億用戶節點及其人際關係邊,規模過於龐大。
- **世界級大型網站**
  - 像是Google、Wikipedia、YouTube等巨型網站,其網頁節點及超連結邊的數量高達數十億甚至上千億,使用BFS探索整個網站可能會資源耗盡。
- **物聯網(IoT)設備網路**
  - 當前許多智慧城市、智慧工廠等領域都會有大量相互連接的IoT設備,形成一個龐大的設備網路,BFS探索整個網路會非常沉重。
- **電信運營商網路**
  - 電信公司維護的通訊網路包含全國各地的交換機、路由器等設備,構成一個超大規模的網路,很難用BFS全面探索。
- **航空公司航線網路**
  - 像是聯合航空、國泰航空等大型航空公司,其全球航點及航線組成了一個十分龐大的網路,使用BFS可能會效率低落。
- **生物分子互作網路**
  - 生物體內存在著蛋白質、基因、代謝物等大量分子及其互作關係,形成一個複雜龐大的分子網路,對BFS來說非常具有挑戰性。