資料結構 C 演算法

圖 Graph

圖 Graph

在一般座標平面上找特定目標真的需要使用到**廣度優先搜尋(BFS)**或**深度優先搜尋(DFS)**嗎？

for 迴圈也可以處理的

```c
#include <stdio.h>
#define SIZE 5
int main()
{
  int i = 0, j = 0;
  int x = 0, y = 0;

  for (i = 0; i < SIZE; i++) {
    x = i;
    printf("[%d, %d]", x, y);
    for (j = 1; j<SIZE; j++) {
      y = (i % 2 == 0) ? y+1:y-1;
      printf("[%d, %d]", x, y);
    }
    printf("\n");
  }
  return 0;
}
```

外層的 for 迴圈
控制 y 坐標值

內層的 for 迴圈
控制 x 軸

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)

執行結果

```
dice - graphSearch $ ./forXY.exe
[0, 0][0, 1][0, 2][0, 3][0, 4]
[1, 4][1, 3][1, 2][1, 1][1, 0]
[2, 0][2, 1][2, 2][2, 3][2, 4]
[3, 4][3, 3][3, 2][3, 1][3, 0]
[4, 0][4, 1][4, 2][4, 3][4, 4]
```

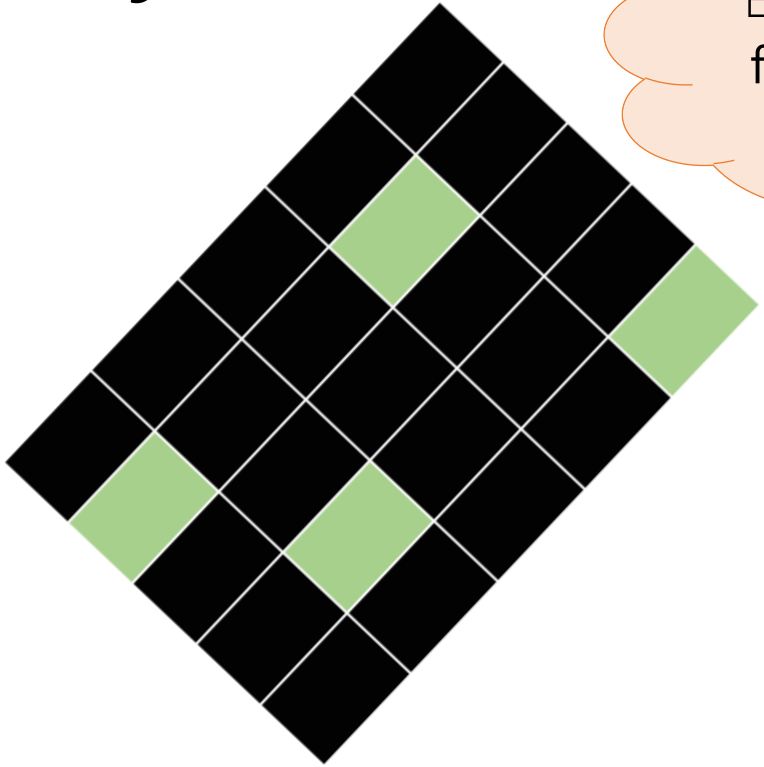
兩個 for 可以解決的事情為什麼還要用廣度優先搜尋(BFS)或深度優先搜尋(DFS)呢？

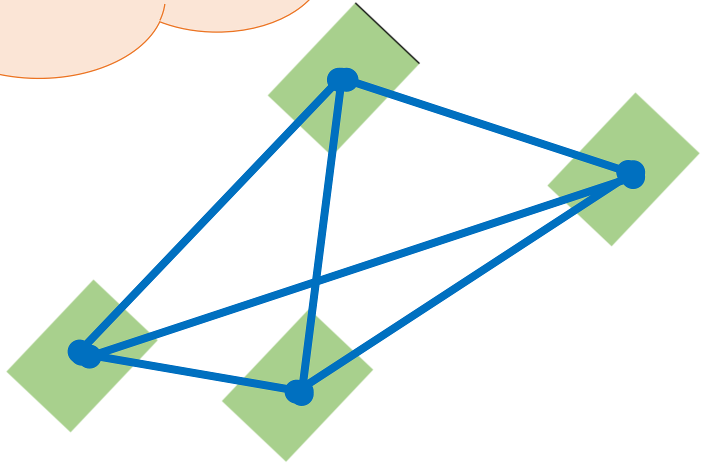
(0,0)	(0,1)	(0,2)	(0,3)	(0,4)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)

```
for (i = 0; i < SIZE; i++) {
  x = i;
  for (j = 1; j<SIZE; j++) {
    y = (i % 2 == 0) ? y+1:y-1;
  }
}
```

如果是這樣的 xy 平面呢？

當平面不再四四方方，
可使用的點有限制時，
for 迴圈好像就不那麼
好使用了？





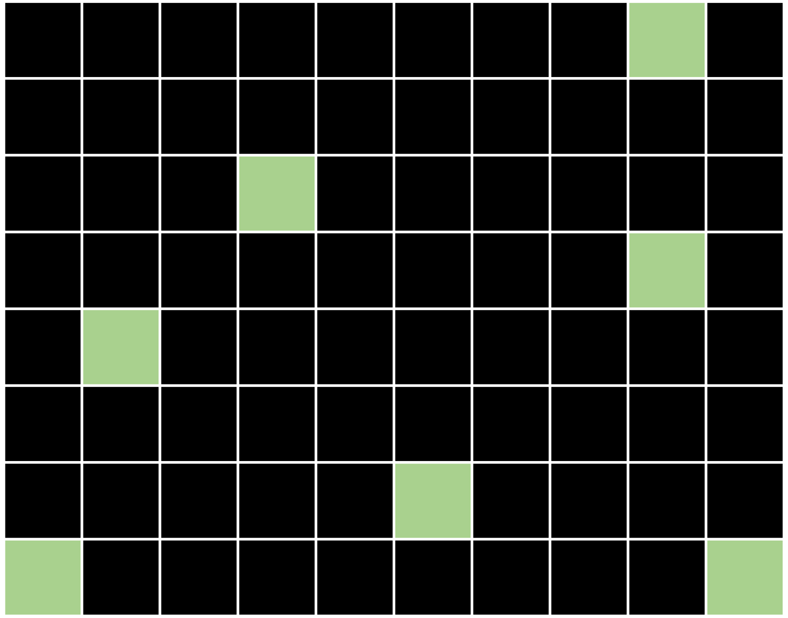
平面上只有
(1,0) (1,3) (3,1) (4,4)
四個點可以使用

將平面旋轉一下

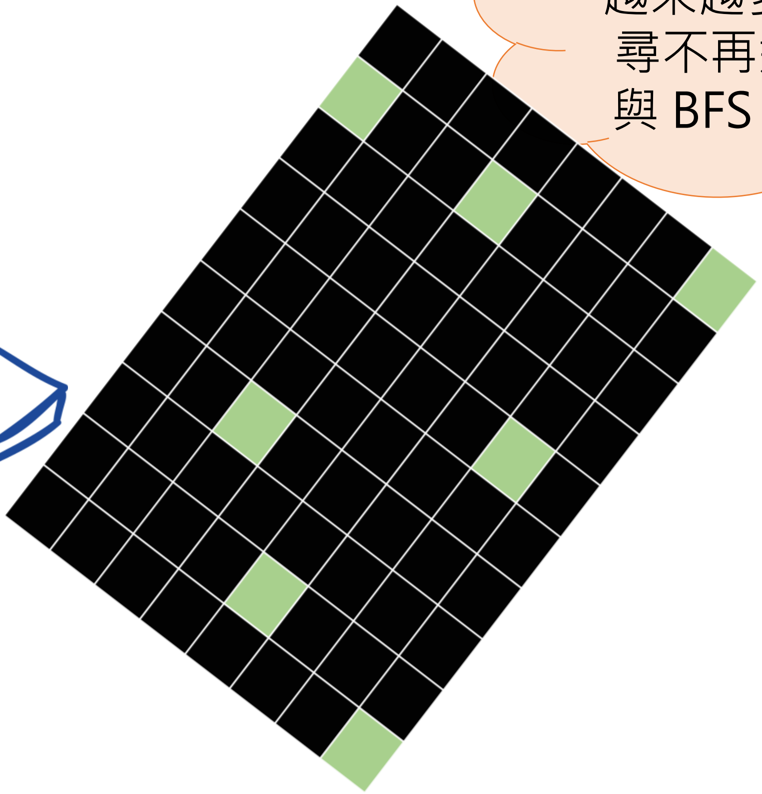
移除不能使用的點，
可以使用的點用線連
接，只能通過線到達
另一個點

甚至是這樣的 xy 平面呢？

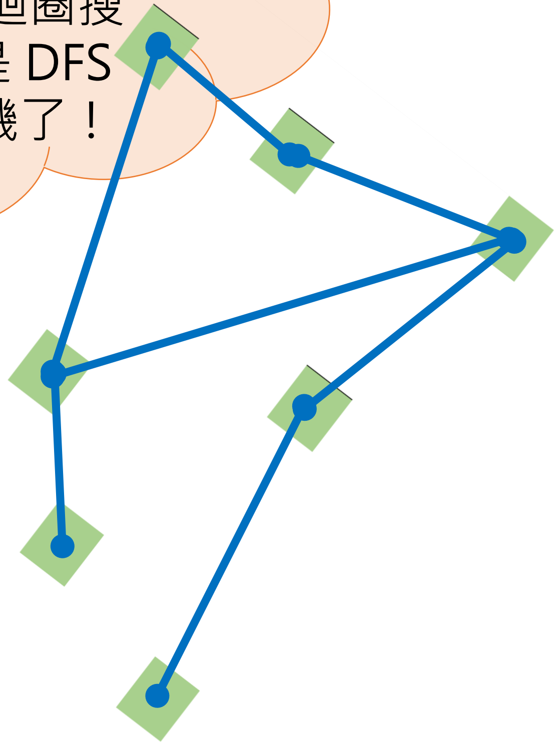
不規則的平面越來越大，
可使用的點與線的限制
越來越多時，for 迴圈搜
尋不再適用，就是 DFS
與 BFS 的使用時機了！



平面長寬變的更大，
可使用與不可使用的
點也都變多了！

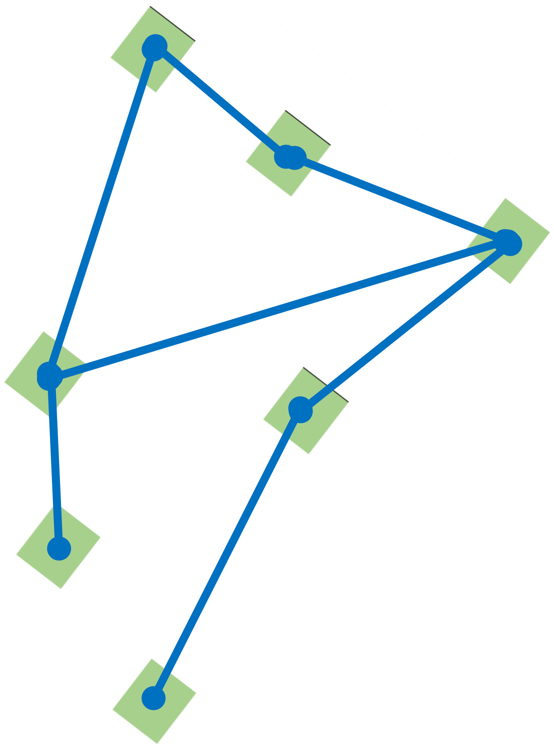


同樣將平面旋轉一下



同樣的，可以通過的點
用線連接，只能通過線
到達另一個點，而且並
非所有點都可互通！

這就是『圖 Graph』

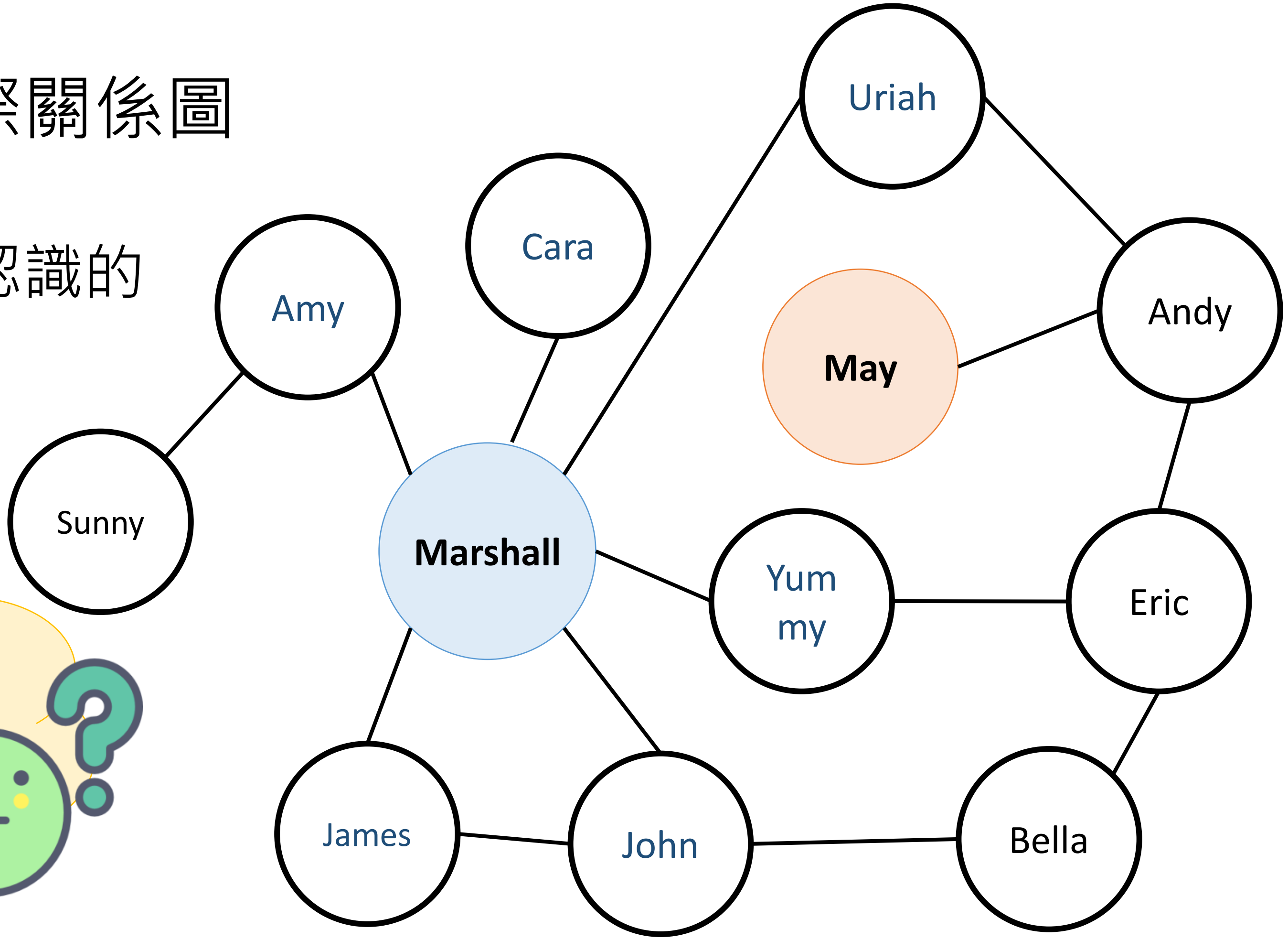


當平面上並非所有點都可使用，可以使用的**點與點**之間必須有**線**連接時才可到達，我們就將此變形平面稱為

『圖Graph』

圖的應用 – 人際關係圖

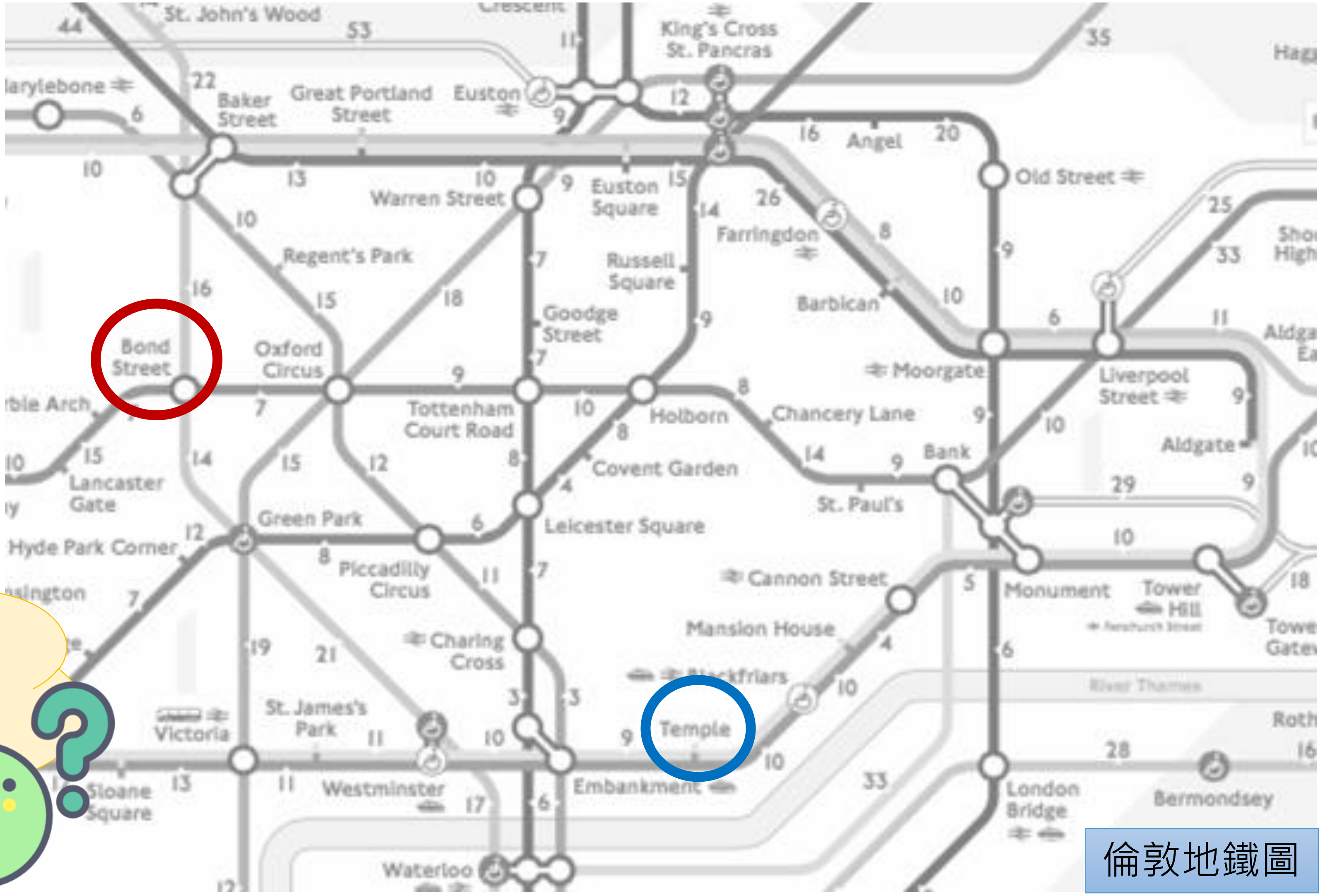
- 圖(graph)也可用來表示人際關係圖
  - 每個點代表一個人
  - 點與點間的線代表兩個人是認識的



想想看，如果 Marshall 想
認識May 的話，可以透過
誰介紹呢？

圖的應用 – 交通路線圖

- 圖(graph)也可用來表示交通路線圖
  - 每個點代表一個地名或停靠點
  - 點與點間的線代表此兩個地點是有交通工具可以到達的

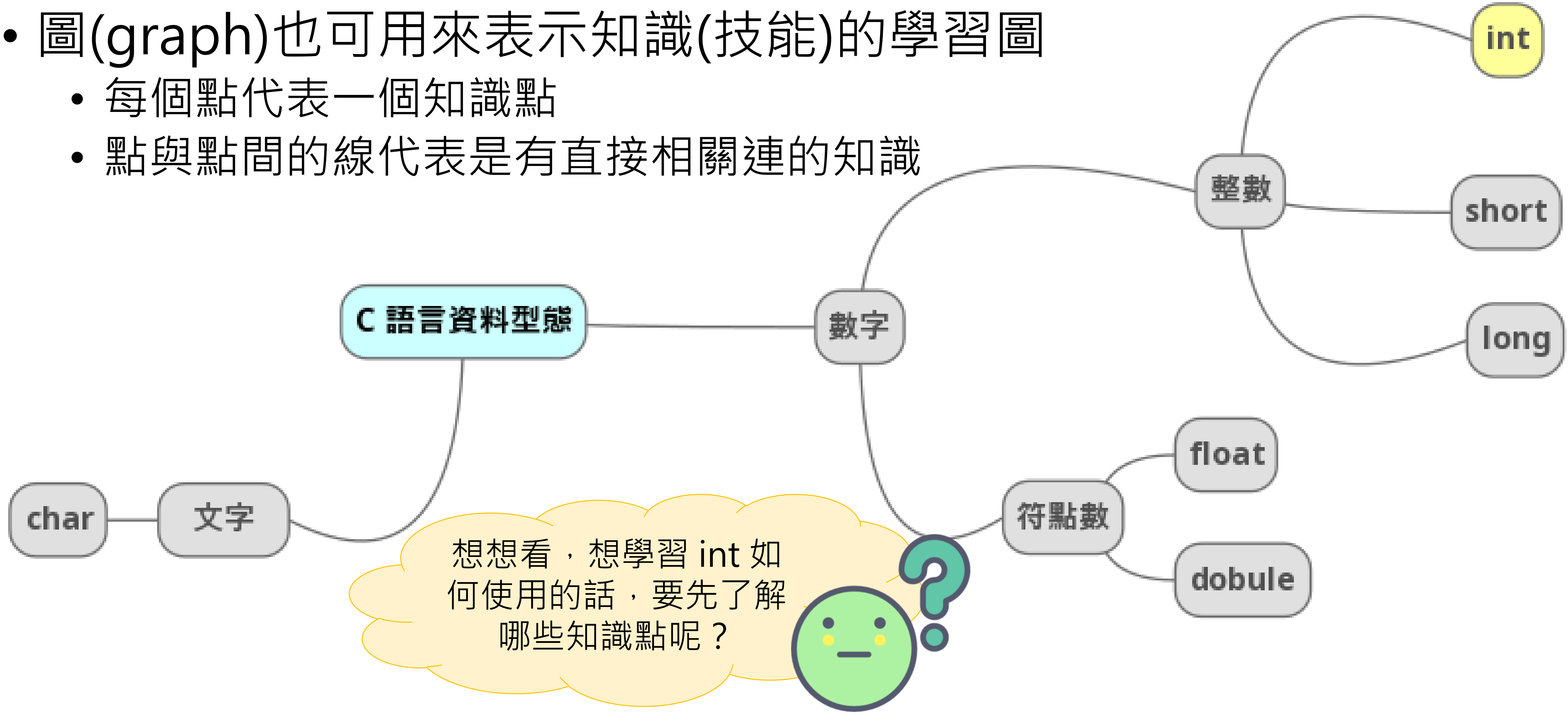


倫敦地鐵圖

想想看，如果想從紅色圈圈的 Bond Street 到達藍色圈圈的 Temple，要如果搭乘呢？

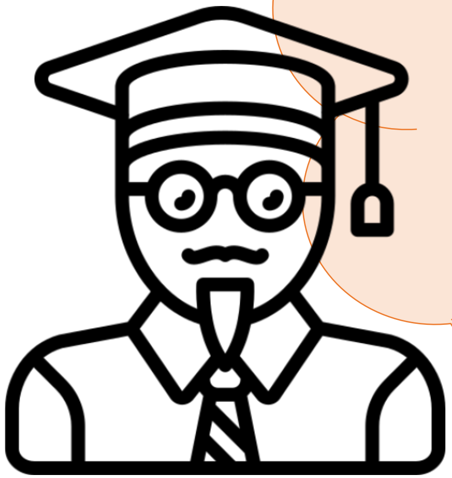
圖的應用 – C 資料型態學習圖

- 圖(graph)也可用來表示知識(技能)的學習圖
  - 每個點代表一個知識點
  - 點與點間的線代表是有直接相關連的知識



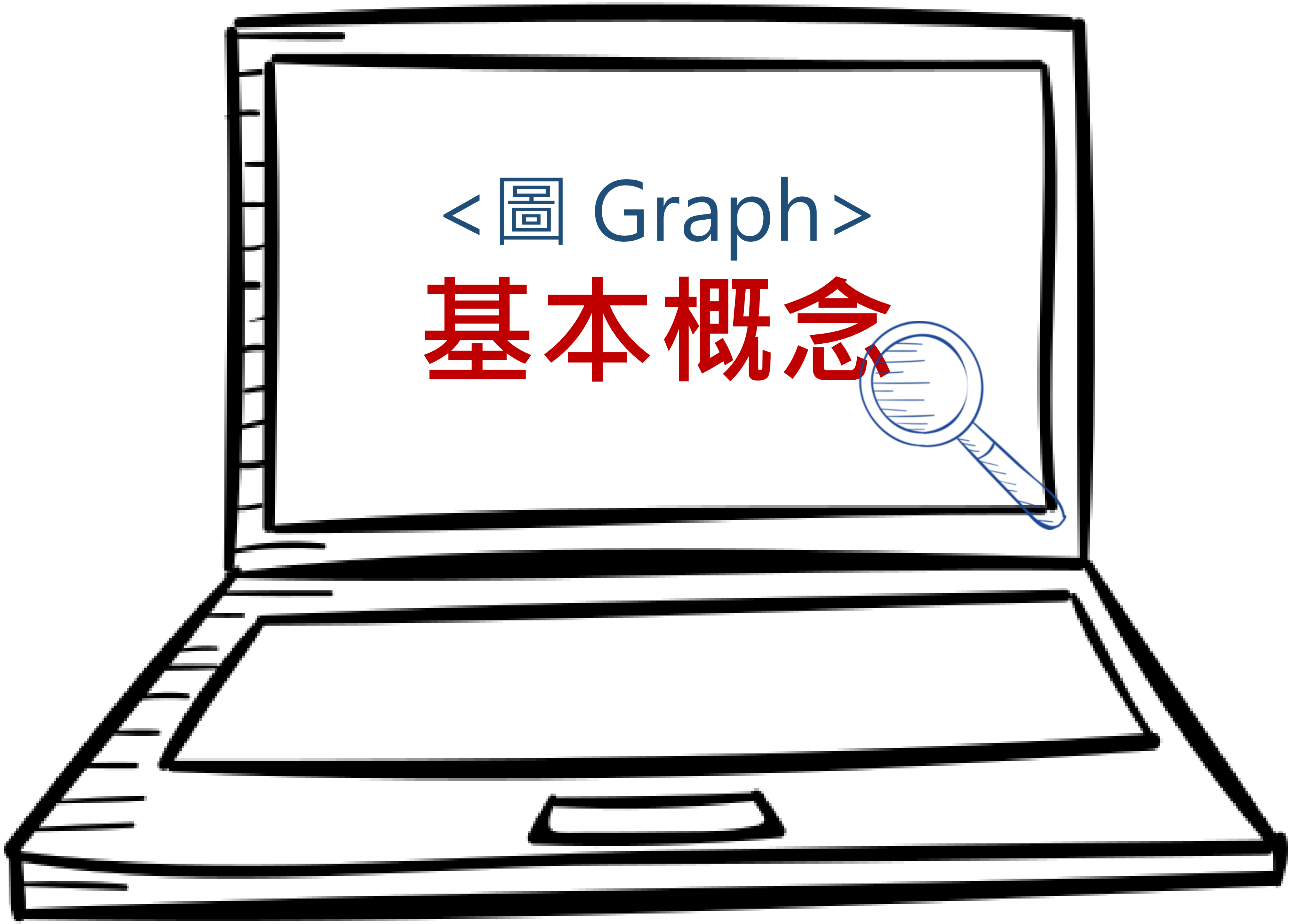
圖的應用 – 很多很多...

- 人際關係圖
- 交通路線圖
- 知識學習圖
- 航線路網
- 電力分布
- 網路分布
- ...



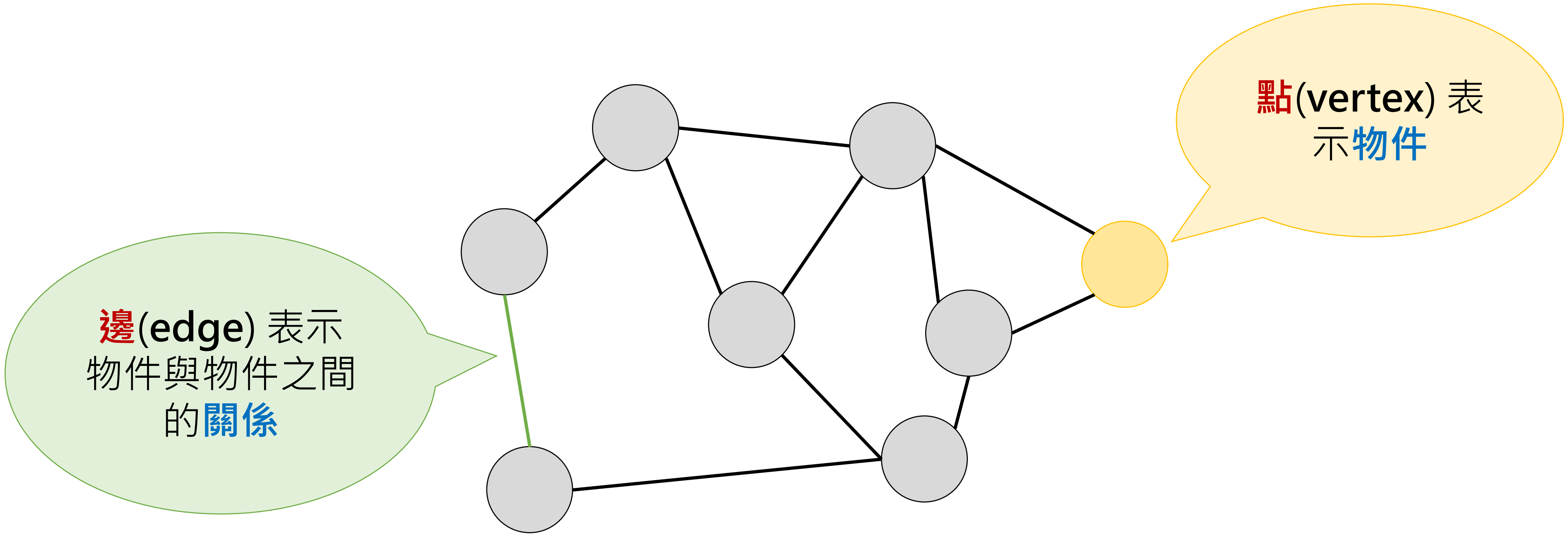
到目前為止，主要介紹為什麼會有『圖 Graph』這種結構，接下來會詳細的介紹『**圖 Graph**』。

因為『圖 Graph』是一種特殊的資料結構，跟前面介紹的指標、結構、排序等等相比有比較多的專有名詞，但不用擔心，不需要硬背名詞，只要搭配圖了解就可以了！


<圖 Graph>
基本概念

圖(Graph) 是？

- 一種表示**物件與物件之間的關係**的資料結構



圖的定義

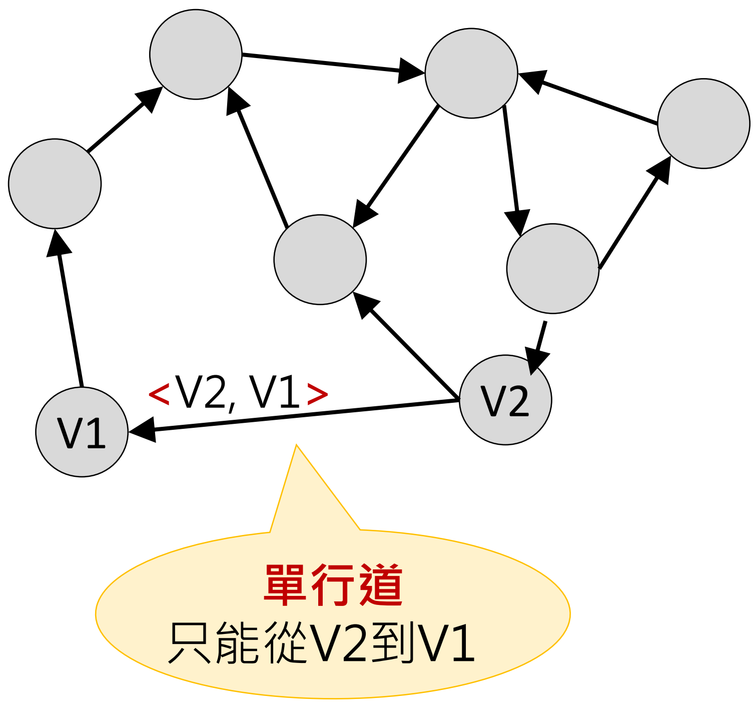
- 圖(Graph)是由**有限的頂點(vertics)**與**有限的邊(edges)**

- 圖的表示法：**G = (V, E)**
  - V 是頂點(verteics)的集合
    - $V(G)=\{V1,V2,V3,\ldots,V_m\}$
    - $m > 0$
  - E 是邊(edges)的集合，任一邊是兩個頂點間的連線，因此用頂點對表示邊
    - $E(G)=\{E1,E2,E3,\ldots,E_n\}$
    - $n > 0$

圖的分類 by 邊的方向性(1)

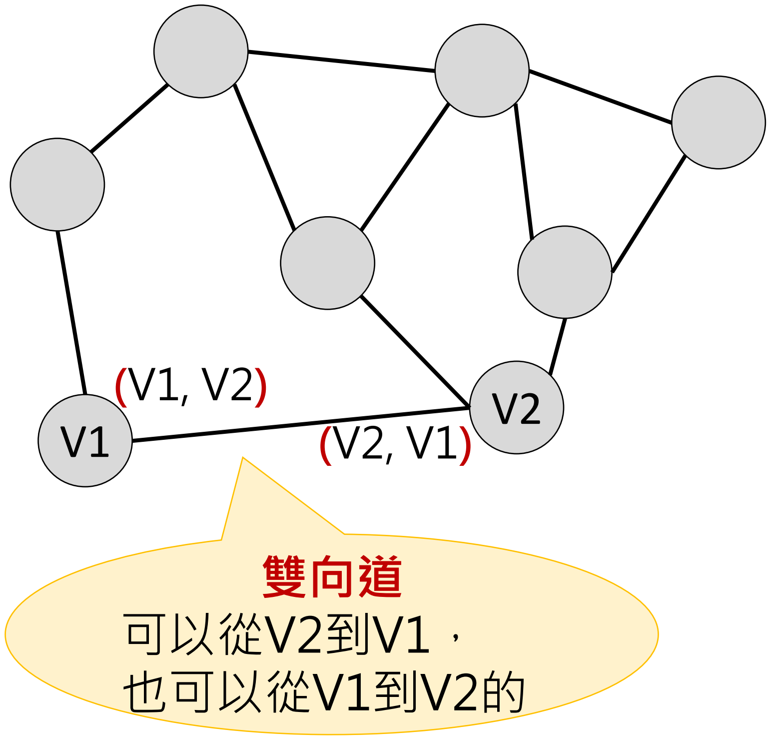
有向圖(directed graph)

- 邊**有**方向性 (**有箭頭**)，箭頭方向會決定頂點的移動方向，



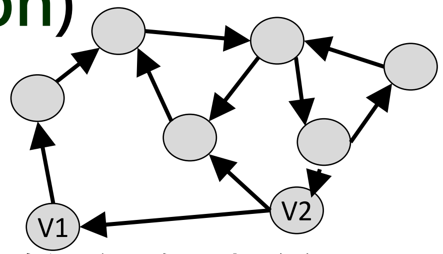
無向圖 (undirected graph)

- 邊**沒有**方向性(**沒有箭頭**)，邊兩端頂點可以互通

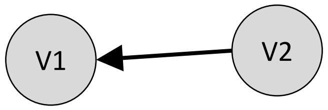


圖的分類 by 邊的方向性(2)

有向圖(directed graph)

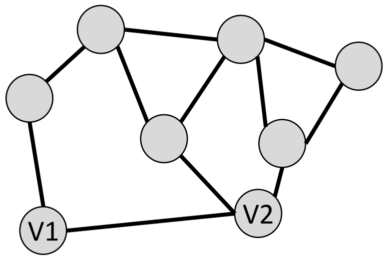


- 邊**有**方向性
- 邊**有**箭頭
- 邊是**單向道**，單向道的方向由箭頭方向決定
- 邊用 **<起始點, 終點>**表示
  - 起始點可以到達終點
  - <V2, V1>: V2可以到達V1
- **<V2, V1> 不等於 <V1, V2>**

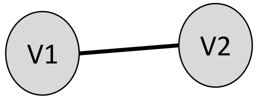


注意！
邊的表示方式是
不同的喔！

無向圖 (undirected graph)



- 邊**沒有**方向性
- 邊**沒有**箭頭
- 任一個邊都是**雙向道**，相接的兩個頂點可以互通
- 邊用 (V2, V1)表示

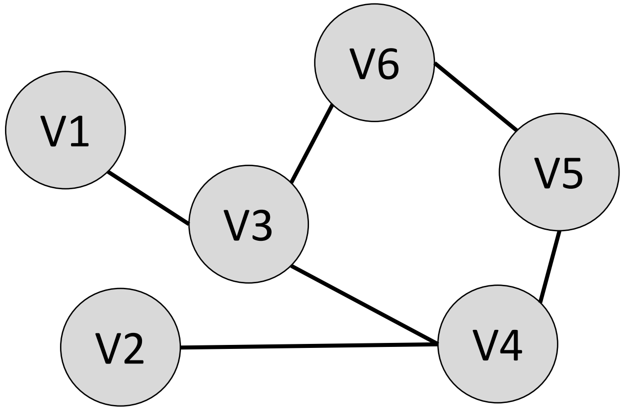


  - 前面那個頂點(V2)可以到達後面那個頂點(V1)，
  - 後面那個頂點(V2)也可以到達前面那個頂點(V 2)
- **(V2, V1) 等於 (V1, V2)**

無向圖的例子

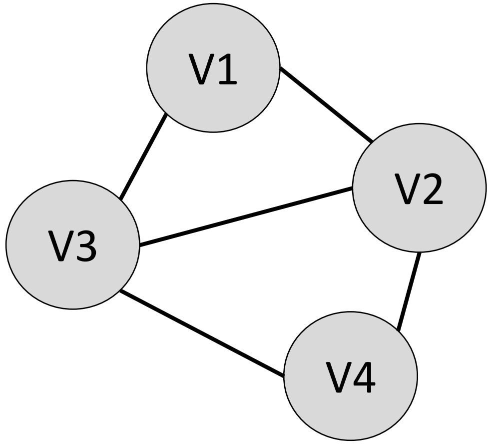
G1

- V(G1) = {V1, V2, V3, V4, V5, V6}
- E(G1) = {(V1, V3), (V3, V1), (V3, V6), (V6, V3), (V5, V6), (V6, V5), (V4, V5), (V5, V4), (V3, V4), (V4, V3), (V2, V4), (V4, V2)}



G2

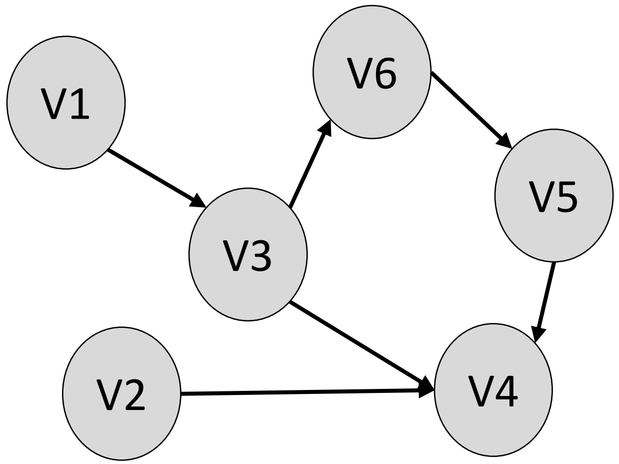
- V(G2) = {V1, V2, V3, V4}
- E(G2) = {(V1, V2), (V2, V1), (V1, V3), (V3, V1), (V2, V3), (V3, V2), (V2, V4), (V4, V2), (V3, V4), (V4, V3)}



有向圖的例子

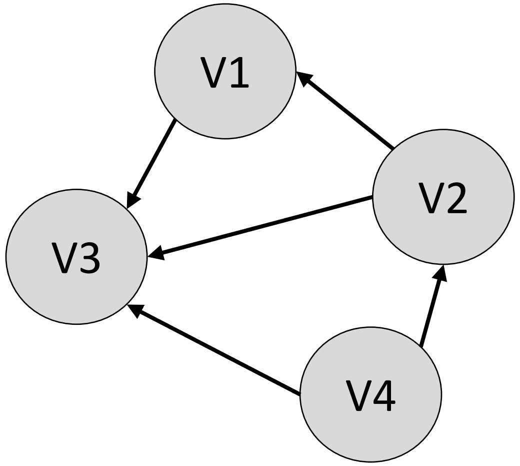
G3

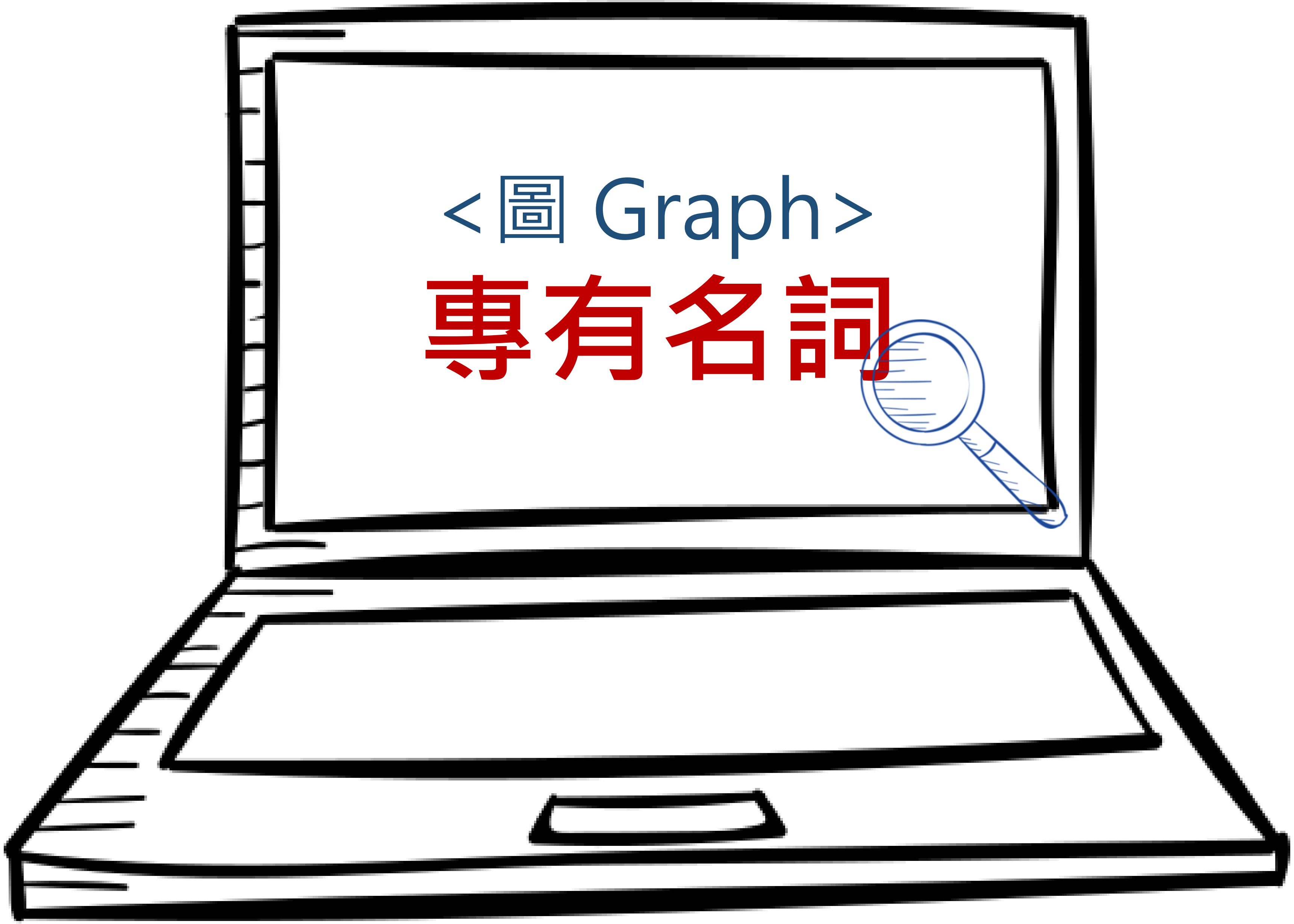
- V(G3) = {V1, V2, V3, V4, V5, V6}
- E(G3) = {<V1, V3>, <V3, V6>, <V6, V5>, <V5, V4>, <V3, V4>, <V2, V4>}



G4

- V(G4) = {V1, V2, V3, V4}
- E(G4) = {<V2, V1>, <V1, V3>, <V2, V3>, <V4, V2>, <V4, V3>}




<圖 Graph>
專有名詞

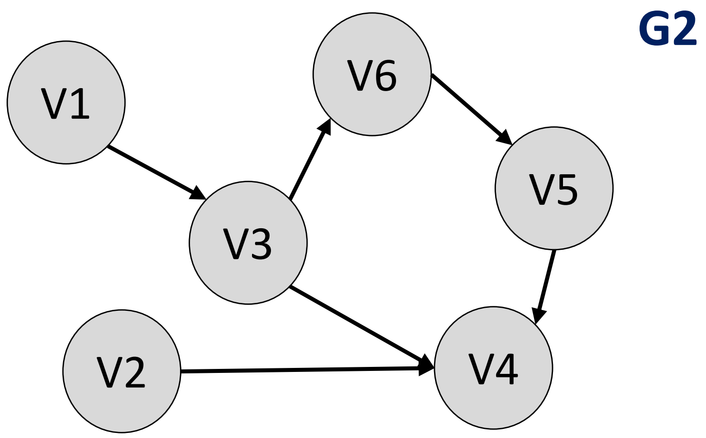
圖的專有名詞

頂點 vertex	相鄰 adjacent	簡單路徑 simple path	緊密連通單元 strongly connected component
邊 edge	附著 incident	循環 cycle	分支度 degree
無向圖 undirected graph	子圖 subgraph	連通 connected	內分支度 in-degree
有向圖 directed graph	路徑 path	連通單元 connected component	外分支度 out-degree
完全圖形 complete graph	長度 length	緊密連通 strongly connected	

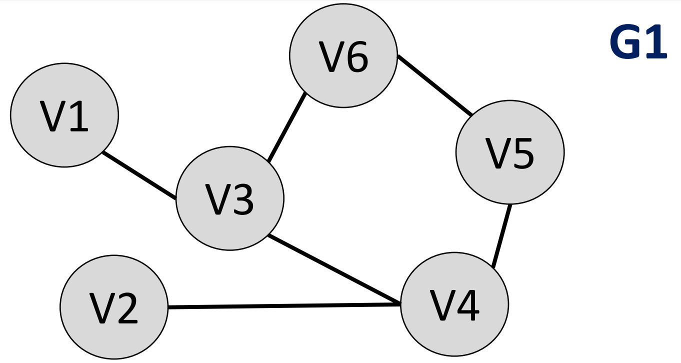
圖的專有名詞

路徑 與 路徑長度

- 路徑(path)：相異兩個頂點之間經過的所有邊稱為路徑
  - 一個路徑是由一個或多個邊組成
  - 相異兩個頂點之間可以有多種路徑
- 路徑長度(path length)：路徑包含的邊的個數



G2中從V1到V5有 1 種路徑：
- 路徑: {<V1, V3>, <V3, V6>, <V6, V5>}
- 路徑長度: 3



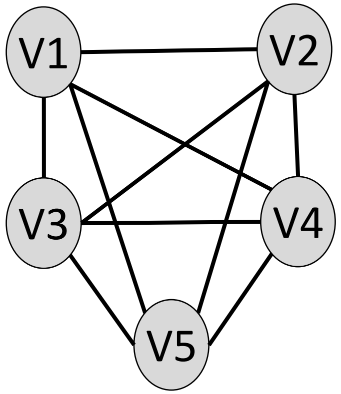
G1 中從V1到V5有 2 種路徑：
- 第 1 種：
  - 路徑: {(V1, V3), (V3, V6), (V6, V5)}
  - 路徑長度: 3
- 第 2 種
  - 路徑2: {(V1, V3), (V3, V4), (V4, V5)}
  - 路徑長度: 3

圖的專有名詞

簡單路徑 與 循環

- 簡單路徑 Simple path
  - 除了起點(第一個頂點)與終點(最後一個頂點)，其他頂點都**不可以重複**

- 循環 Cycle
  - 若一條簡單路徑的**起點與終點是相同的頂點**，就稱為**循環(cycle)**
  - 循環又稱為**迴圈**



- {(V1, V3), (V3, V5), (V5, V2)} 是簡單路徑
- {(V1, V3), (V3, V5), (V5, V4), (V4, V2)} 是簡單路徑
- {(V1, V3), (V3, V5), (V5, V4), (V4, V2), (V2, V5)} **不是**簡單路徑
  - V5 重複出現
- {(V1, V3), (V3, V5), (V5, V1)} 是簡單路徑也是循環

相鄰 與 附著

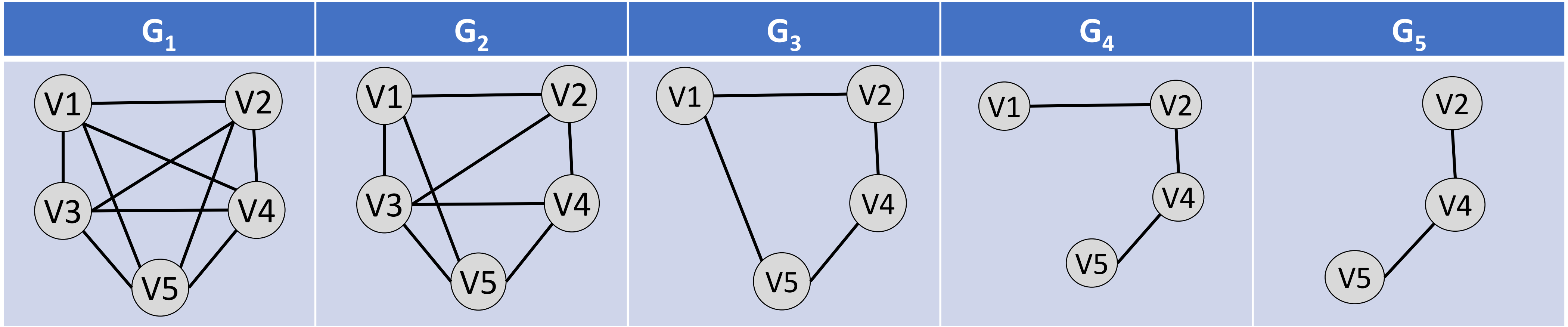
- 相鄰(adjacent)
  - 無向圖：任兩個頂點間有一條邊相接，就稱這兩個頂點是相鄰的。
    - 邊 (V1, V2) 代表 V1 與 V2 是相鄰的。
  - 有向圖：
    - 邊<V1, V2>代表：
      - V1 相鄰至(adjacent to) V2
      - V2 相鄰自(adjacent from)V1

- 附著(incident)
  - 我們稱頂點V_1和頂點V_2是相鄰，而邊 (V_1, V_2) 是附著在頂點V_1與V_2頂點上。

圖的專有名詞

子圖 subgraph

- 若一個圖 G_1 內的全部頂點與邊都含在另一個圖 G_2 內，則稱 G_1 為 G_2 的子圖
- 子圖 G_1 的頂點數與邊數必小於或等於 G_2

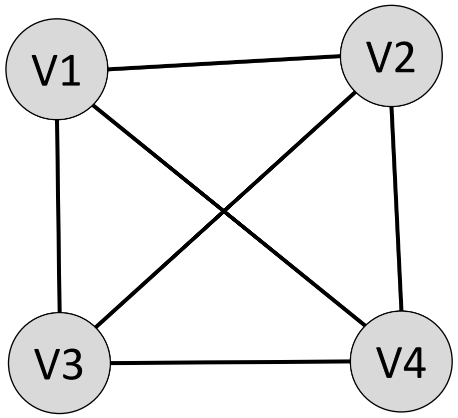
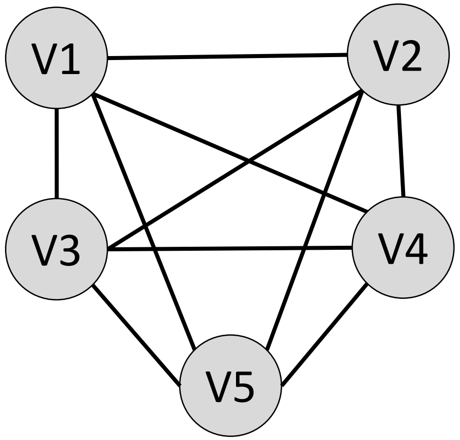


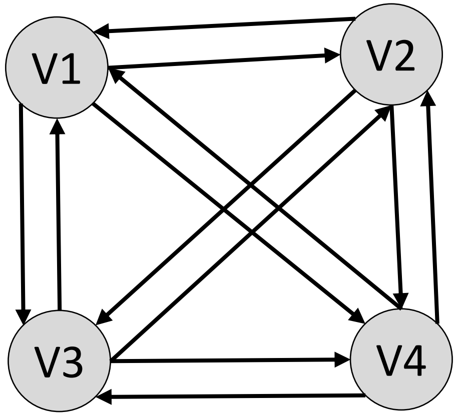
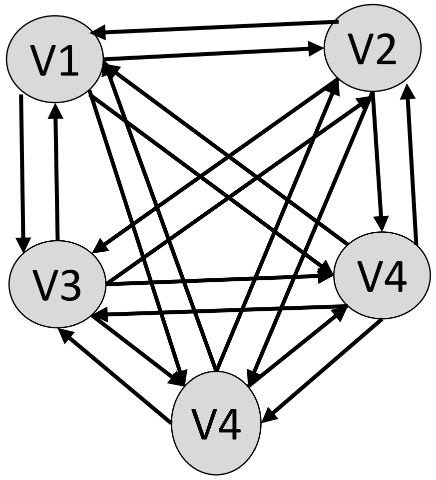
- G_5 是 G_4 的子圖，也是 G_3 的子圖，也是 G_2 的子圖，也是 G_1 的子圖
- G_4 是 G_3 的子圖，也是 G_2 的子圖，也是 G_1 的子圖
- G_3 是 G_2 的子圖，也是 G_1 的子圖
- G_2 是 G_1 的子圖

圖的專有名詞

完全圖 complete graph

- 任兩個**頂點之間都有一條邊**相接的圖稱為**完全圖 complete graph**

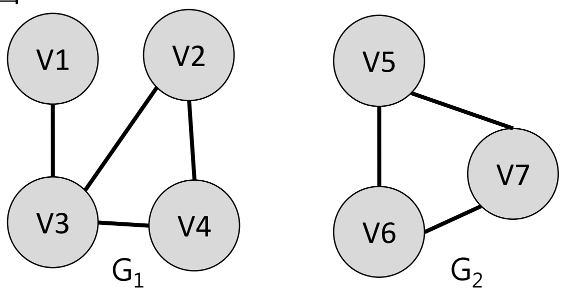
無向完全圖	
N 個頂點的無向完全圖，會有 n(n-1)/2 的邊	
4 個頂點，有6個邊	5 個頂點，有10個邊
	

有向完全圖	
N 個頂點的有向完全圖，會有 n(n-1) 的邊	
4 個頂點，有12個邊	5 個頂點，有20個邊
	

連通

- 連通 connected
  - 若兩個頂點之間存在路徑，稱此兩點為連通的。
- 連通圖 connected graph
  - 圖形G中，任兩點都有路徑存在，就稱圖形G為連通圖。
- 連通單元 connected component
  - 圖形 G 中最大的連通子圖。
  - 當圖 G_1 是另一個圖 G_2 的子圖時，且 G_1 是連通圖時，以及 G_1 只能是 G_2 的連通子圖時，G_1 就是 G_2 的連通單元。
  - 一個圖可能有多個連通單元。

圖G

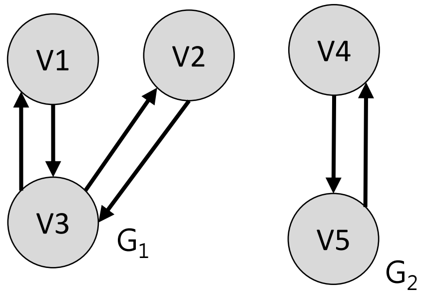


- G1 與 G2 是圖G的子圖
- V(G)={V1,V2,V3,V4,V5,V6,V7}
- V1 與 V4 是連通的
- V1 與 V6 不連通
- G1 與 G2 都是連通圖，也是圖G的連通子圖，也是圖G的連通單元

緊密連通

- 緊密連通 strongly connected
  - 若**有向圖**G中任兩個不同頂點，皆存在至少一條可以互通到對方頂點的路徑，就稱此圖G為緊密連通
- 緊密連通單元strongly connected component
  - **有向圖**內的緊密連通最大子圖

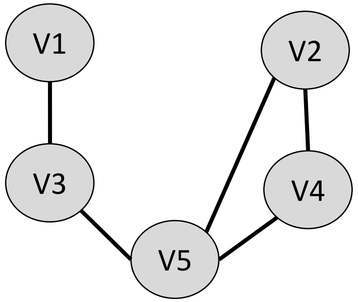
圖G



•G1 與 G2 是圖G的子圖
•V(G)={V1,V2,V3,V4,V5}
•G1 與 G2 都是緊密連通
•G1 與 G2是G的緊密連通單元

分支度 degree

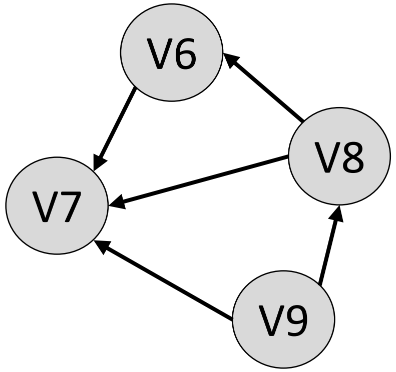
- 分支度 degree
  - 附著在頂點的邊數。
  - 只有**無向圖**會記算頂點的分支度。



- V1分支度: 1
- V2分支度: 2
- V3分支度: 2
- V4分支度: 2
- V5分支度: 3

- 內分支度 in-degree
  - 頂點V的內分支度是指以V為終點（即箭頭指向V）的邊數。
  - 只有**有向圖**會計算頂點的內分支度。

- 外分支度 out-degree
  - 頂點V的外分支度是指以V為起點的邊數。
  - 只有**有向圖**會計算頂點的外分支度。

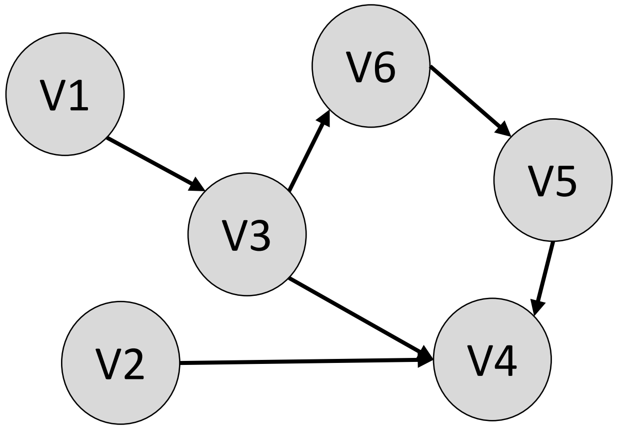


- V6內分支度: 1
  外分支度: 1
- V7內分支度: 3
  外分支度: 0
- V8內分支度: 1
  外分支度: 2
- V9內分支度: 0
  外分支度: 2

同構圖 Graph Isomorphism

- 當兩個圖的邊數與節點數相同，且兩個圖用 V(G) 與 E(G) 表示時，節點與邊的表示都一樣時，就稱此兩個圖為**同構**

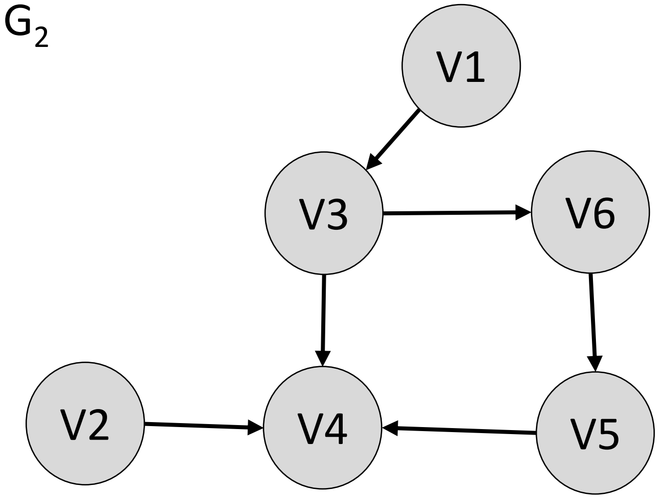
圖G_1



- $V(G_1)$ = {V1, V2, V3, V4, V5, V6}
- $E(G_1)$ = {<V1, V3>, <V3, V6>, <V6, V5>, <V5, V4>, <V3, V4>, <V2, V4>}

G1與G2是同構的

圖G_2



- $V(G_2)$ = {V1, V2, V3, V4, V5, V6}
- $E(G_2)$ = {<V1, V3>, <V3, V6>, <V6, V5>, <V5, V4>, <V3, V4>, <V2, V4>}

圖的專有名詞

不通用的名詞

僅適用於無向圖的名詞

- 分支度(degree)

僅適用於有向圖的名詞

- 緊密連通 strongly connected
- 緊密連通單元strongly connected component
- 內分支度 in-degree
- 外分支度 out-degree

專有名詞表

頂點 vertex	相鄰 adjacent	簡單路徑 simple path	緊密連通單元 strongly connected component
邊 edge	附著 incident	循環 cycle	分支度 degree
無向圖 undirected graph	子圖 subgraph	連通 connected	內分支度 in-degree
有向圖 directed graph	路徑 path	連通單元 connected component	外分支度 out-degree
完全圖形 complete graph	長度 length	緊密連通 strongly connected	

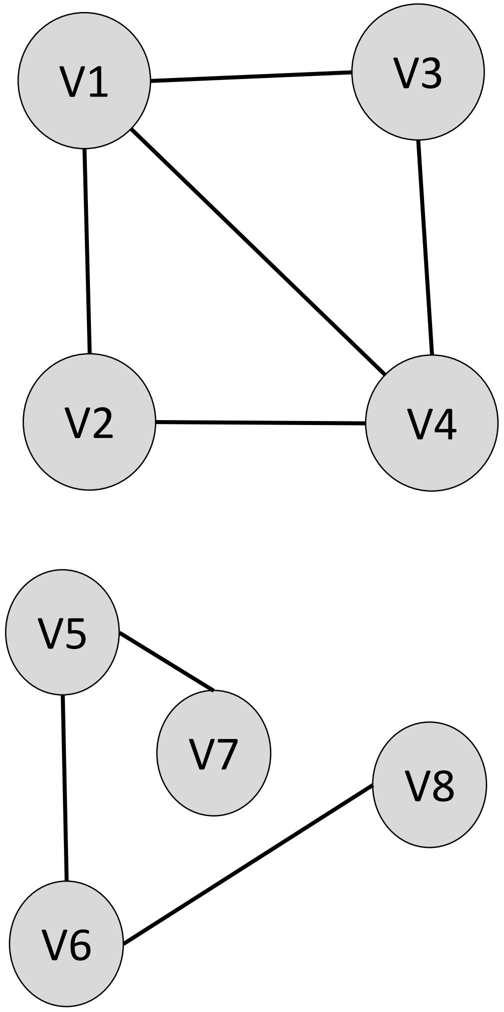
試著每看到一個圖形就將上面的專有名詞表逐一填入對應的值，多練習幾次，相信很快就會熟記了喔！

下面舉出兩種練習例子，趕快來試試看吧！

圖的名詞練習 G_1

- G_1是無向圖或有向圖？
- $V(G_1)=\{?\}$
- $E(G_1)=\{?\}$
- G_1是完全圖形嗎？
- V1與V2有相鄰嗎？V1與V4有相鄰嗎？V1與V8有相鄰嗎？
- 有哪些邊附著在V1？有哪些邊附著在V5？
- 請畫出三種G_1的子圖
- 請列出V5到V8的路徑與路徑長度，並說明列出的是簡單路徑嗎？
- 請列出一條路徑長度為4，起始點為V1的循環路徑
- 請列出G_1的連通單元
- 請說明G_1內各頂點的分支度

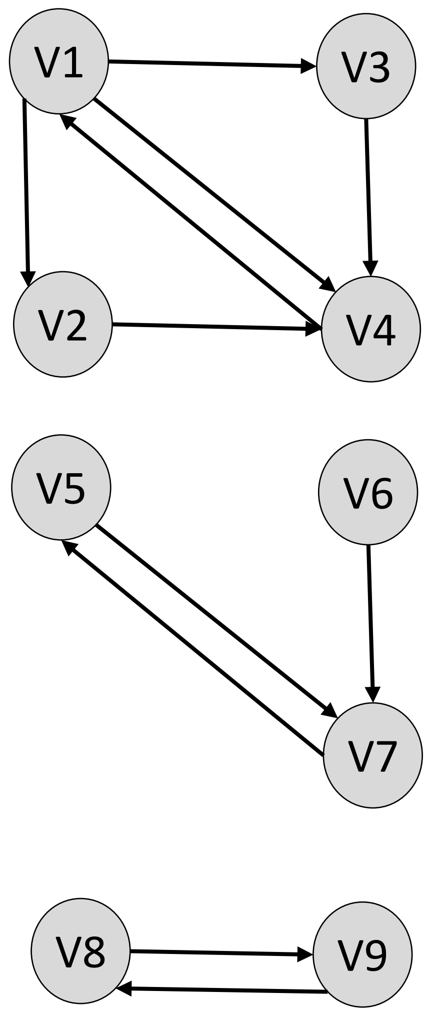
圖G_1



圖的名詞練習 G_2

- G_2是無向圖或有向圖？
- $V(G_2)=\{?\}$
- $E(G_2)=\{?\}$
- G_2是完全圖形嗎？
- V1相鄰至哪些點？V1相鄰自哪些點？
- 有哪些邊附著在V1？有哪些邊附著在V5？
- 請畫出三種G_2的子圖
- 請列出V1到V4的路徑與路徑長度，並說明列出的是簡單路徑嗎？
- 請列出 2 條路徑長度為3，起始點為V1的循環路徑
- 請列出G_1的連通單元與緊密連通單元
- 請說明G_1內各頂點的內分支度與外分支度

圖G_2

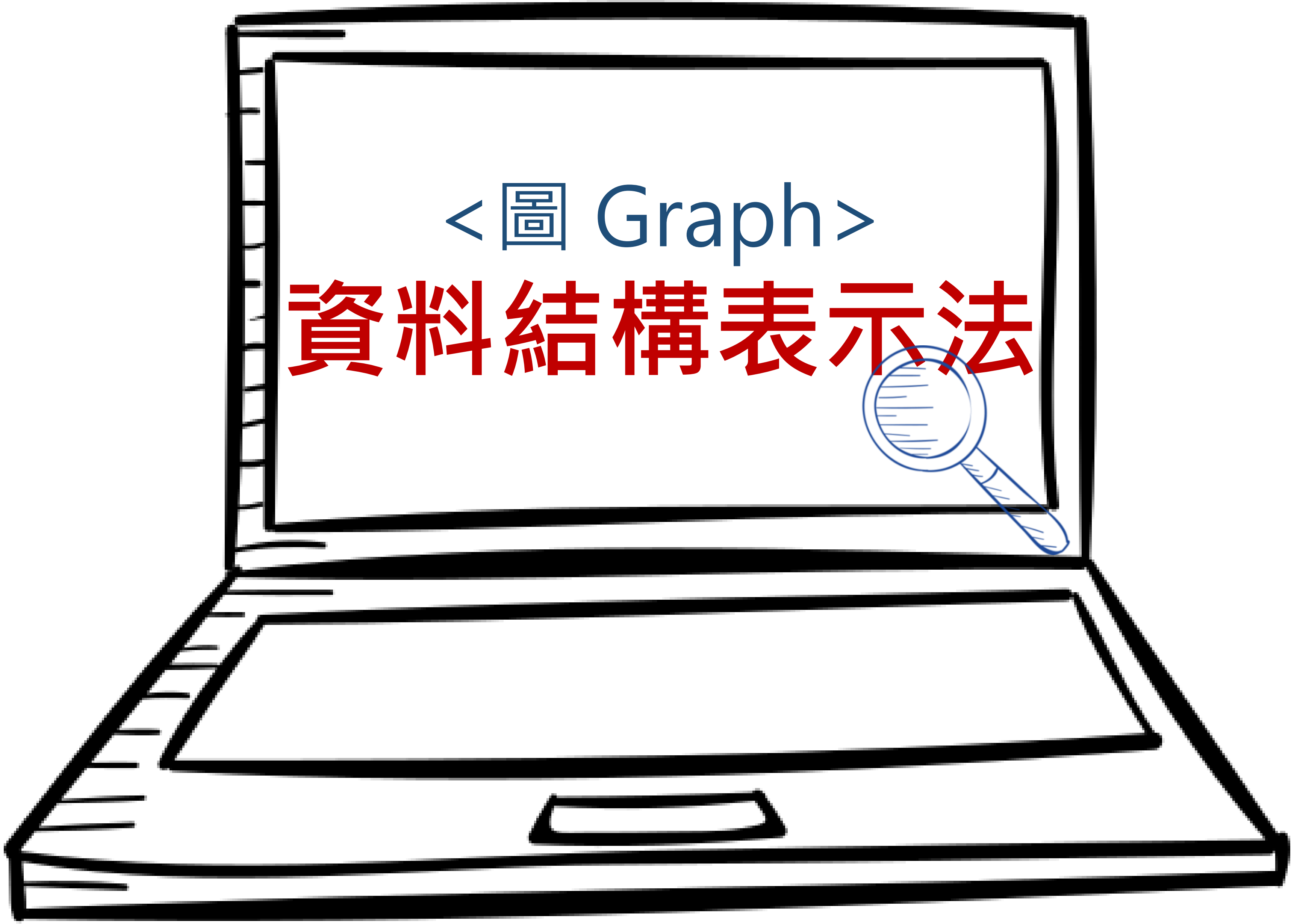


中途休息一下，
先回想我們現在要學習什麼？

- 我們最終目標是
  - 處理圖的問題

- 目前我們
  - 瞭解了什麼是圖 Graph
  - 瞭解了圖 Graph實際有哪些應用
  - 瞭解了圖有哪些種類與名詞

- 那我們還需要瞭解什麼呢？
  - 要先能**用資料結構表示圖形**，才能用程式處理圖的問題啊！

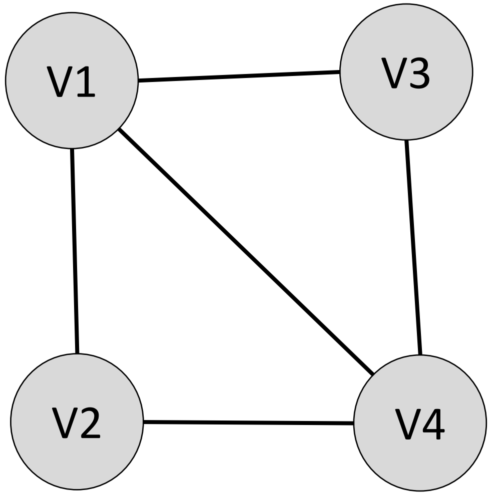

<圖 Graph>
資料結構表示法

如何表示圖形結構？

- 相鄰矩陣(adjacency matrix)
  - 使用二維陣列
- 相鄰串列(adjacency list)
  - 使用鏈節串列

相鄰矩陣 adjacency matrix

- 使用一個 **N * N 的二維整數陣列**儲存圖形結構，稱為**相鄰矩陣**
  - N 是圖形的頂點數目
  - 陣列內每個元素(V_i, V_j)表示這兩個頂點之間是否有邊，有邊元素值就是1，沒有邊元素值就是0，因此元素值只有 **0 或 1** 兩種值
  - 對角線上的元素值衡為 0
    - 目前處理的圖都是簡單圖(simple graph)，不會有自我迴路(self loop)的情況，因此對角線上的元素值都會是 0
    - 簡單圖的介紹可參考後面的延伸觀念1



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

製作無向圖的相鄰矩陣 (1)

步驟 1: 使用頂點數目決定陣列大小

- 5個頂點，整數陣列大小就是 5 * 5

	1	2	3	4	5
1					
2					
3					
4					
5					

步驟 2: 將所有陣列元素值初始化為 0

	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0

製作無向圖的相鄰矩陣 (2)

步驟 3: 關注第 1 列，設定 (V_1, V_j) 的值

- V_1 代表第一個頂點
- V_j 代表第 j 個頂點
- 若 V_1 與 V_j 相鄰，就將(V_1, V_j)設為 1，不相鄰就維持元素值為 0

	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0

步驟 4: 重複步驟3，每次關注一列(一個頂點 i)，設定該列的所有元素值(V_i, V_j)，值到每列設定完畢

- 關注第 2 列，設定(V_2, V_j)的值
- 關注第 3 列，設定(V_3, V_j)的值
- 關注第 4 列，設定(V_4, V_j)的值
- 關注第 5 列，設定(V_5, V_j)的值

	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0

製作有向圖的相鄰矩陣 (1)

步驟 1: 使用頂點數目決定陣列大小

- 5個頂點，整數陣列大小就是 5 * 5

	1	2	3	4	5
1					
2					
3					
4					
5					

步驟 2: 將所有陣列元素值初始化為 0

	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0

製作有向圖的相鄰矩陣 (2)

步驟 3: 關注第 1 列，設定 (V_1, V_j) 的值

- V_1 代表第一個頂點
- V_j 代表第 j 個頂點
- 若 V_1 有一條箭頭指向V_j的邊，就將(V_1, V_j)設為 1，沒有的話就維持元素值為 0

	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0

步驟 4: 重複步驟3，每次關注一列(一個頂點 i)，設定該列的所有元素值(V_i, V_j)，值到每列設定完畢

- 關注第 2 列，設定(V_2, V_j)的值
- 關注第 3 列，設定(V_3, V_j)的值
- 關注第 4 列，設定(V_4, V_j)的值
- 關注第 5 列，設定(V_5, V_j)的值

	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0

相鄰矩陣 adjacency matrix

無向圖

- 有 M 個邊就有 M*2 個元素值為 1
- 第 I 列有幾個元素值為1就代表第 I 個頂點的**分支度**為多少，也代表有多少邊附著在第 I 個頂點
- 第 J 行有幾個元素值為1就代表第 J 個頂點的分支度為多少，也代表有多少邊附著在第 J 個頂點

有向圖

- 有 M 個邊就有 M 個元素值為 1
- 第 I **列**有幾個元素值為1就代表第 I 個頂點的**外分支度**為多少
- 第 J **行**有幾個元素值為1就代表第 J 個頂點的**內分支度**為多少

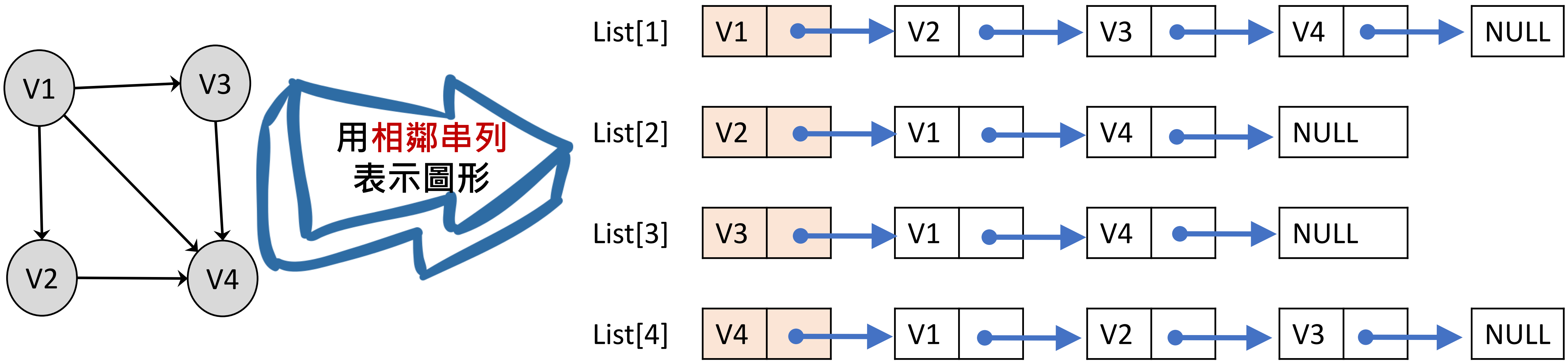
	1	2	3	4
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0

第 2 列 =>第 2 個頂點的外分支度

第 3 行 =>第 3 個頂點的內分支度

相鄰串列(adjacency list)

- 使用 **N 個單向鏈節串列**串接每個頂點的相鄰頂點
  - N 是圖形的頂點數目
  - 第 I 個單向鏈節串列代表附著在第 I 個頂點的邊



製作無向圖的相鄰串列 (1)

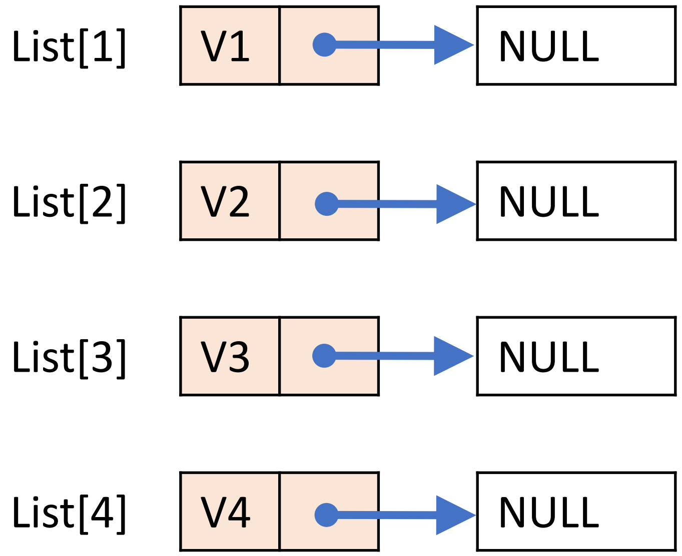
步驟 1: 定義單向鏈節串列節點結構

- 結構內至少包含兩個成員：
  - 是第幾個頂點
  - 指向下一個節點的結構指標

```c
typedef struct vnode {
  int vertex;
  struct vnode *next;
} graphNode;
```

步驟 2: 使用頂點數目決定結構指標陣列大小，並初始化陣列元素

- 4個頂點，指標陣列大小就是 4
- 並將每個元素的下一個節點指標設為 NULL

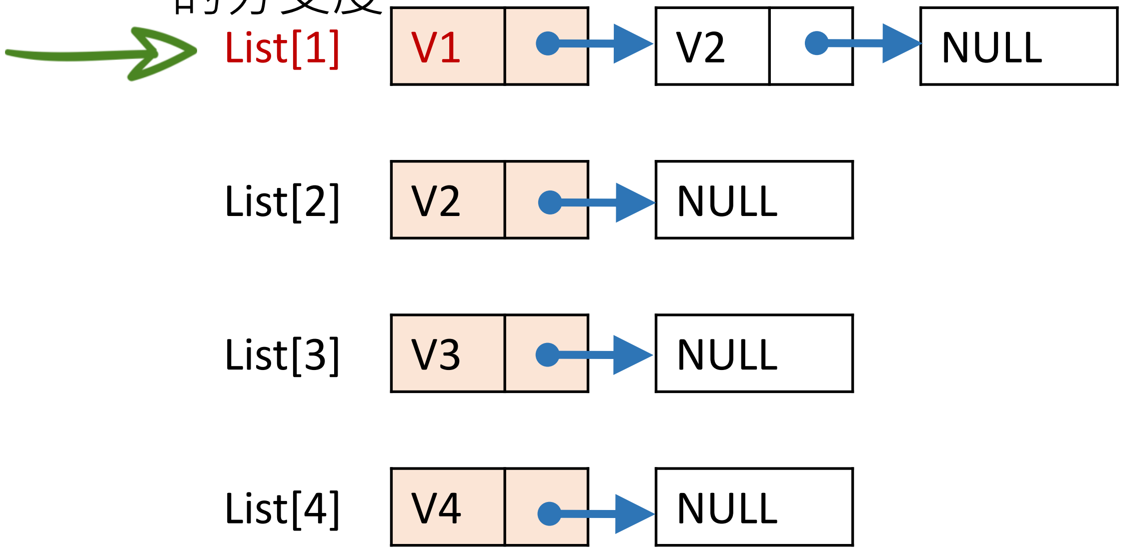



```c
graphNode *graphVertex[4];
```

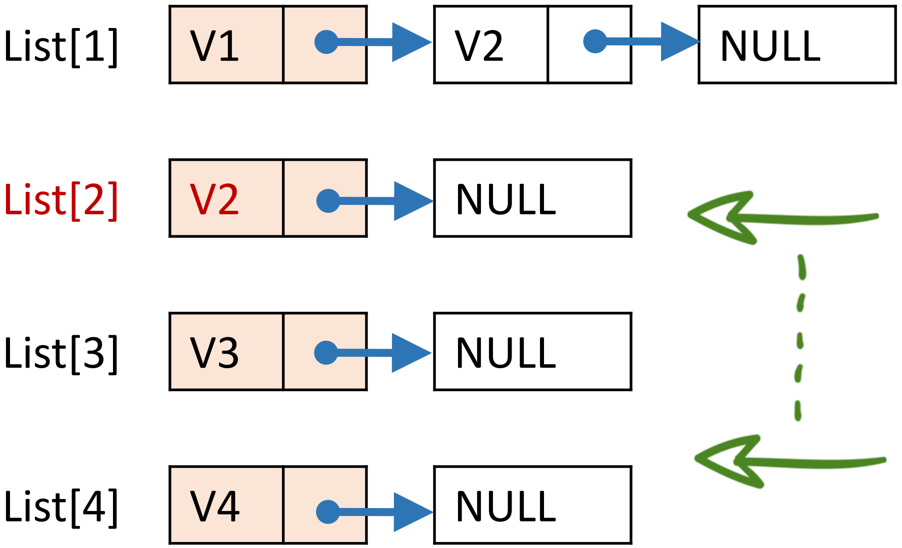
製作無向圖的相鄰串列 (2)

步驟 3: 關注結構陣列的第 1 個項目，將跟第 1 個頂點相鄰的頂點加到第 1 個串列列表內

- 邊 (v1, v4) 會在第 1 個串列加入頂點 4，也會在第 4 個串列加入節點 1
- 第 I 個鏈節串列的節點數會等於第 I 個頂點的分支度



步驟 4: 重複步驟3，每次一個陣列項目，設定與該對應頂點的相鄰點



製作有向圖的相鄰串列 (1)

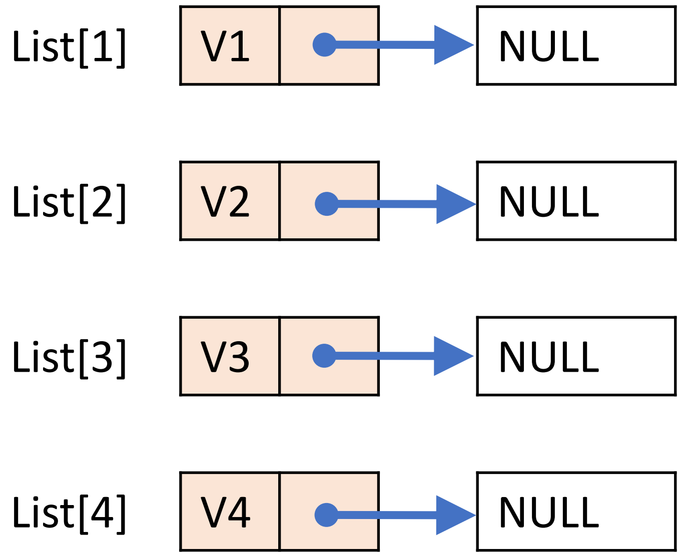
步驟 1: 定義單向鏈節串列節點結構

- 結構內至少包含兩個成員：
  - 是第幾個頂點
  - 指向下一個節點的結構指標

```
typedef struct vnode {
  int vertex;
  struct vnode *next;
} graphNode;
```

步驟 2: 使用頂點數目決定結構指標陣列大小，並初始化陣列元素

- 4個頂點，指標陣列大小就是 4
- 並將每個元素的下一個節點指標設為 NULL

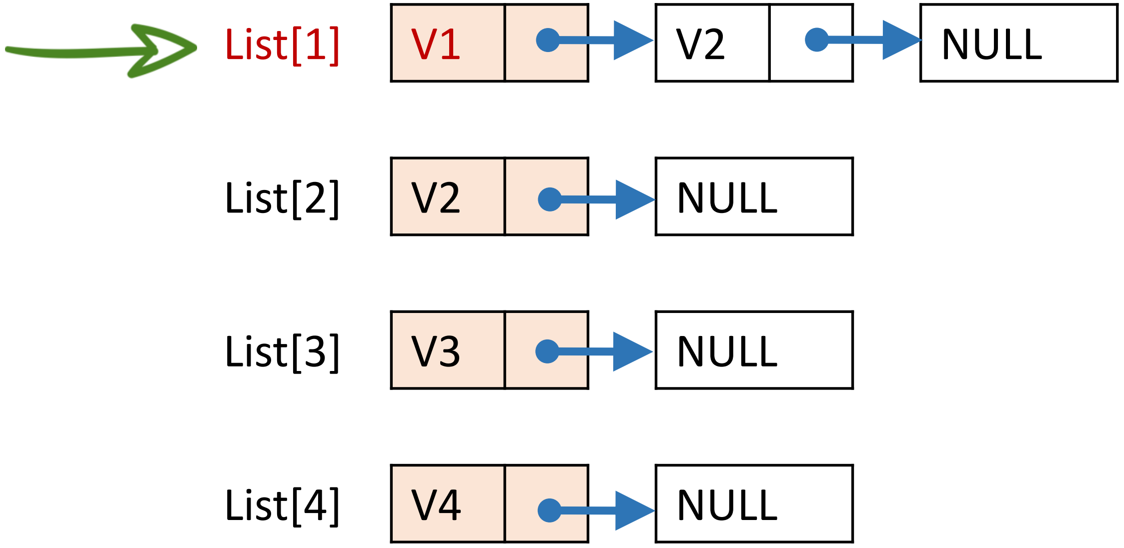



```
graphNode *graphVertex[4];
```

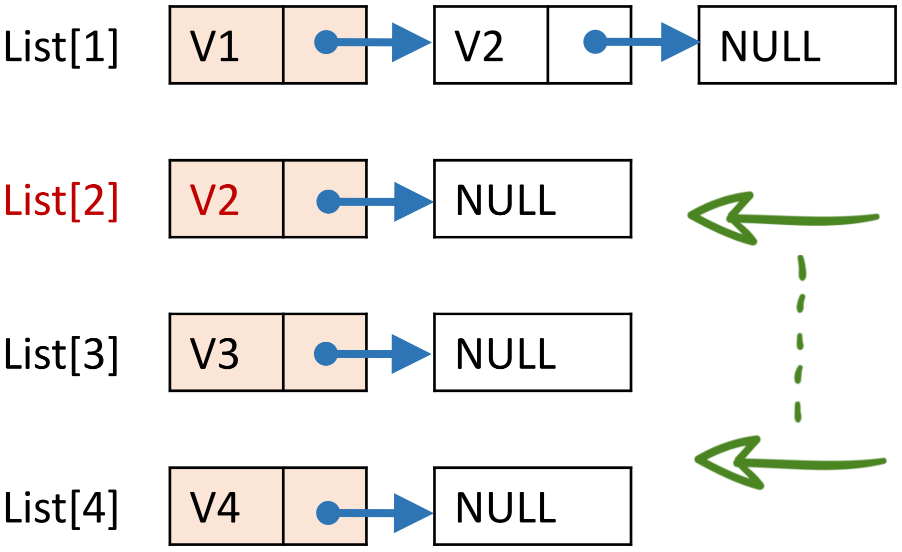
製作有向圖的相鄰串列 (2)

步驟 3: 關注結構陣列的第 1 個項目，將從第 1 個頂點指出去的邊對應的相鄰頂點加到第 1 個串列列表內

- 邊 <v1, v4> 只會在第 1 個串列加入頂點 4
- 第 I 個鏈節串列的節點數會等於第 I 個頂點的外分支度



步驟 4: 重複步驟3，每次一個陣列項目，設定與該對應頂點的相鄰點



	相鄰矩陣	相鄰串列
	1. 使用二維陣列 2. 頂點數決定整數陣列大小	1. 使用鏈節串列 2. 頂點數決定結構指標陣列大小
無向圖	1. 對稱矩陣 2. 有 M 個邊就有 M*2 個元素值為 1 3. 第 I 列有幾個元素值為1就代表第 I 個頂點的分支度為多少，也代表有多少邊附著在第 I 個頂點 4. 第 J 行有幾個元素值為1就代表第 J 個頂點的分支度為多少，也代表有多少邊附著在第 J 個頂點	1. 有 M 個邊就有 M*2 個節點 2. 第 I 個串列有幾個節點就代表第 I 個頂點的分支度為多少，也代表有多少邊附著在第 I 個頂點
有向圖	1. 有 M 個邊就有 M 個元素值為 1 2. 第 I 列有幾個元素值為1就代表第 I 個頂點的外分支度為多少 3. 第 J 行有幾個元素值為1就代表第 J 個頂點的內分支度為多少	1. 有 M 個邊就有 M 個節點 2. 第 I 個串列有幾個節點就代表第 I 個頂點的外分支度為多少

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

第 2 列 => 第 2 個頂點的外分支度

第 3 行 => 第 3 個頂點的內分支度

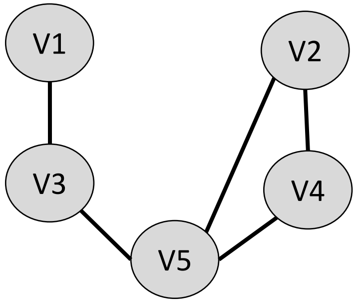
	相鄰矩陣	相鄰串列
	1. 使用**二維陣列** 2. 頂點數決定整數陣列大小	1. 使用**鏈節串列** 2. 頂點數決定結構指標陣列大小
無向圖	1. 對稱矩陣 2. 有 M 個邊就有 M*2 個元素值為 1 3. 第 I 列有幾個元素值為1就代表第 I 個頂點的分支度為多少，也代表有多少邊附著在第 I 個頂點 4. 第 J 行有幾個元素值為1就代表第 J 個頂點的分支度為多少，也代表有多少邊附著在第 J 個頂點	1. 有 M 個邊就有 M*2 個節點 2. 第 I 個串列有幾個節點就代表第 I 個頂點的分支度為多少，也代表有多少邊附著在第 I 個頂點
有向圖	1. 有 M 個邊就有 M 個元素值為 1 2. 第 I 列有幾個元素值為1就代表第 I 個頂點的**外**分支度為多少 3. 第 J 行有幾個元素值為1就代表第 J 個頂點的**內**分支度為多少	1. 有 M 個邊就有 M 個節點 2. 第 I 個串列有幾個節點就代表第 I 個頂點的**外**分支度為多少

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

第 2 列 =>第 2 個頂點的外分支度

第 3 行 =>第 3 個頂點的內分支度

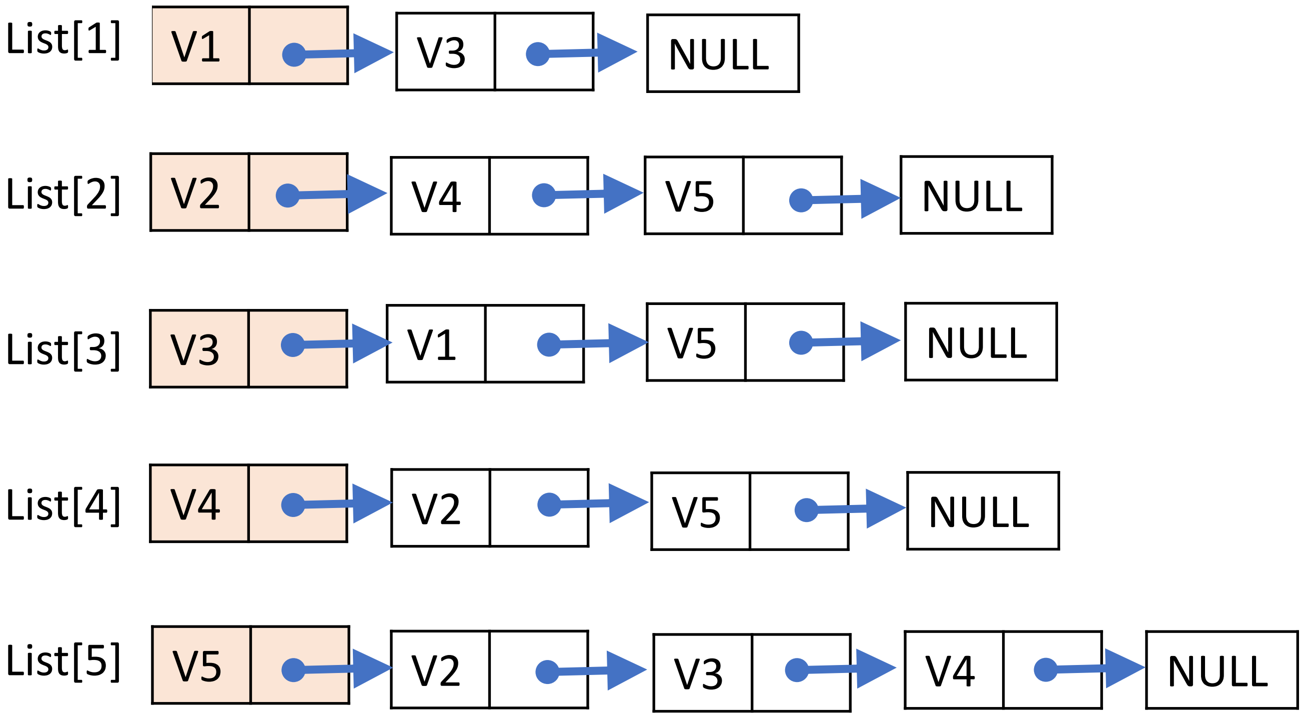
圖形的表示例子(1)



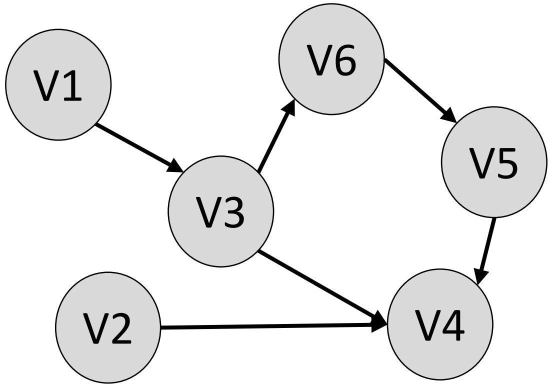
相鄰矩陣

	1	2	3	4	5
1	0	0	1	0	0
2	0	0	0	1	1
3	1	0	0	0	1
4	0	1	0	0	1
5	0	1	1	1	0

相鄰串列



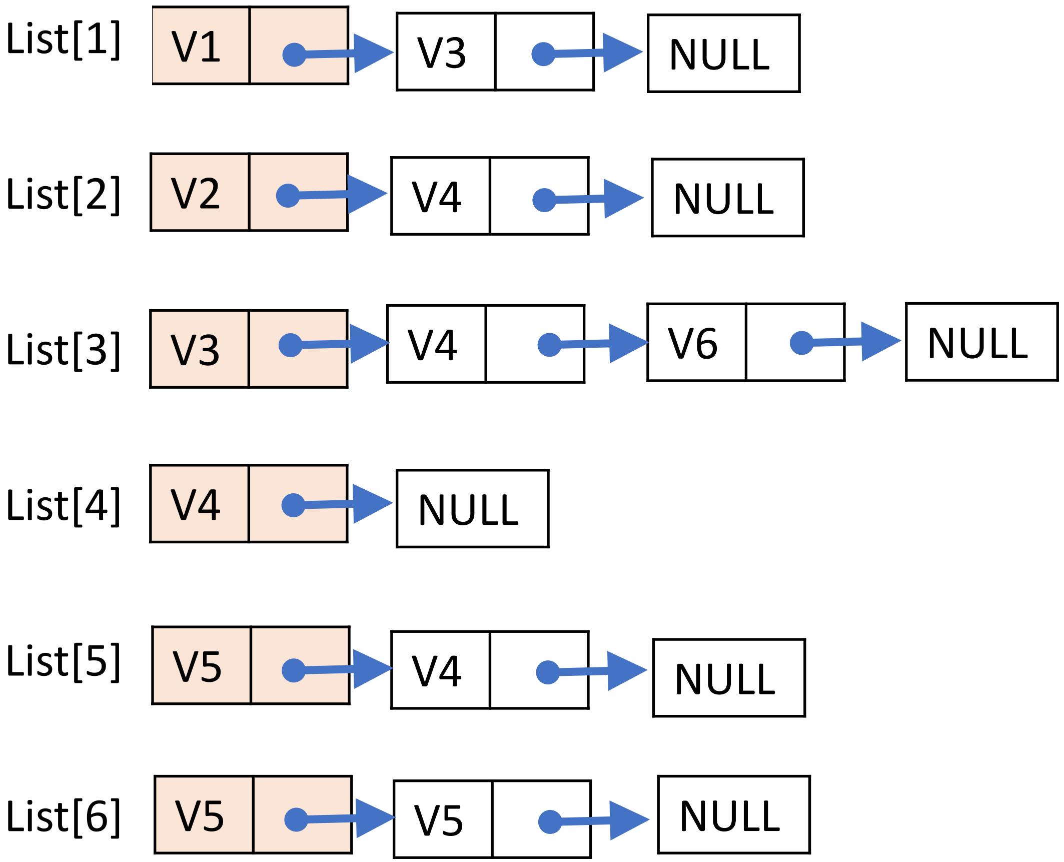
圖形的表示例子(2)



相鄰矩陣

	1	2	3	4	5	6
1	0	0	1	0	0	0
2	0	0	0	1	0	0
3	0	0	0	1	0	1
4	0	0	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	1	0

相鄰串列



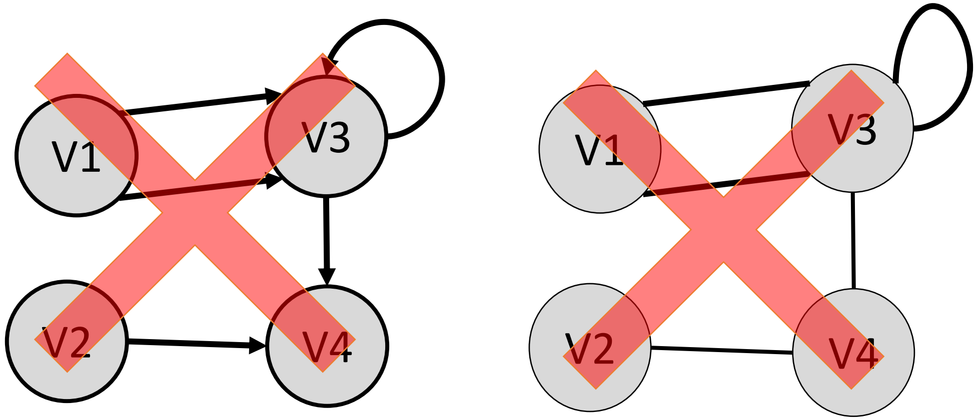
相鄰矩陣 V.S.相鄰串列

	相鄰矩陣	相鄰串列
實作	比較容易（只需要整數陣列）	比較麻煩（需要使用指標、結構、鏈節串列）
判斷任兩點是否有邊	簡單，只要取 V[i][j] 的值，若是 1 就是有邊	麻煩，需要將指定頂點對應的串列列表檢查一遍
邊數遠少於頂點數	浪費空間	省空間
邊數與頂點數差異不大	省空間	相對浪費空間（需要多儲存下一個節點的指標）
邊數不多時，計算總邊數	耗時（要用兩層迴圈檢視所有元素值）	省時（節點相對少）

延伸的概念

概念1: 簡單圖 Simple Graph

- 沒有多重邊(multiple edges)
- 沒有自我迴路(self loop)



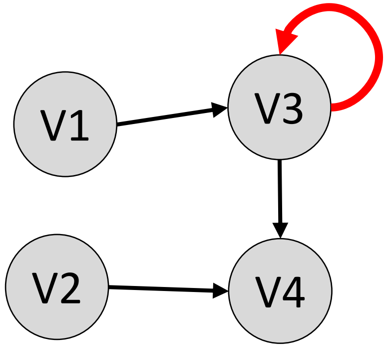
一般我們探討的圖形問題都是使用簡單圖

概念1: 簡單圖 Simple Graph

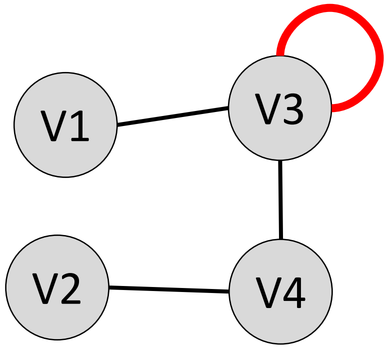
沒有自我迴路(self loop)

- 邊的**兩端點**都**是同一個頂點**的邊稱為 self loop

有向圖內的自我迴路



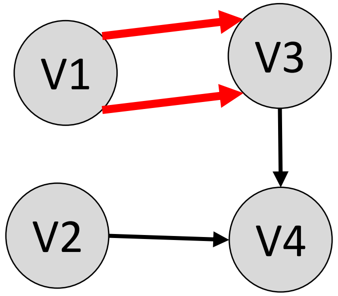
無向圖內的自我迴路



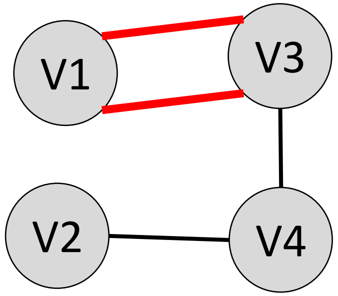
沒有多重邊(multiple edges)

- 兩條**起點與終點都相同**的邊稱為多重邊，也稱為**平行邊(parallel edge)**
- 有多重邊的圖稱為**多重圖(multigraph)**
- 兩個頂點(V1 與 V2)之間包含的多重邊的邊數撐為邊(V1, V2)或(或<V1, V2>)的**重數**

有向圖內的多重邊

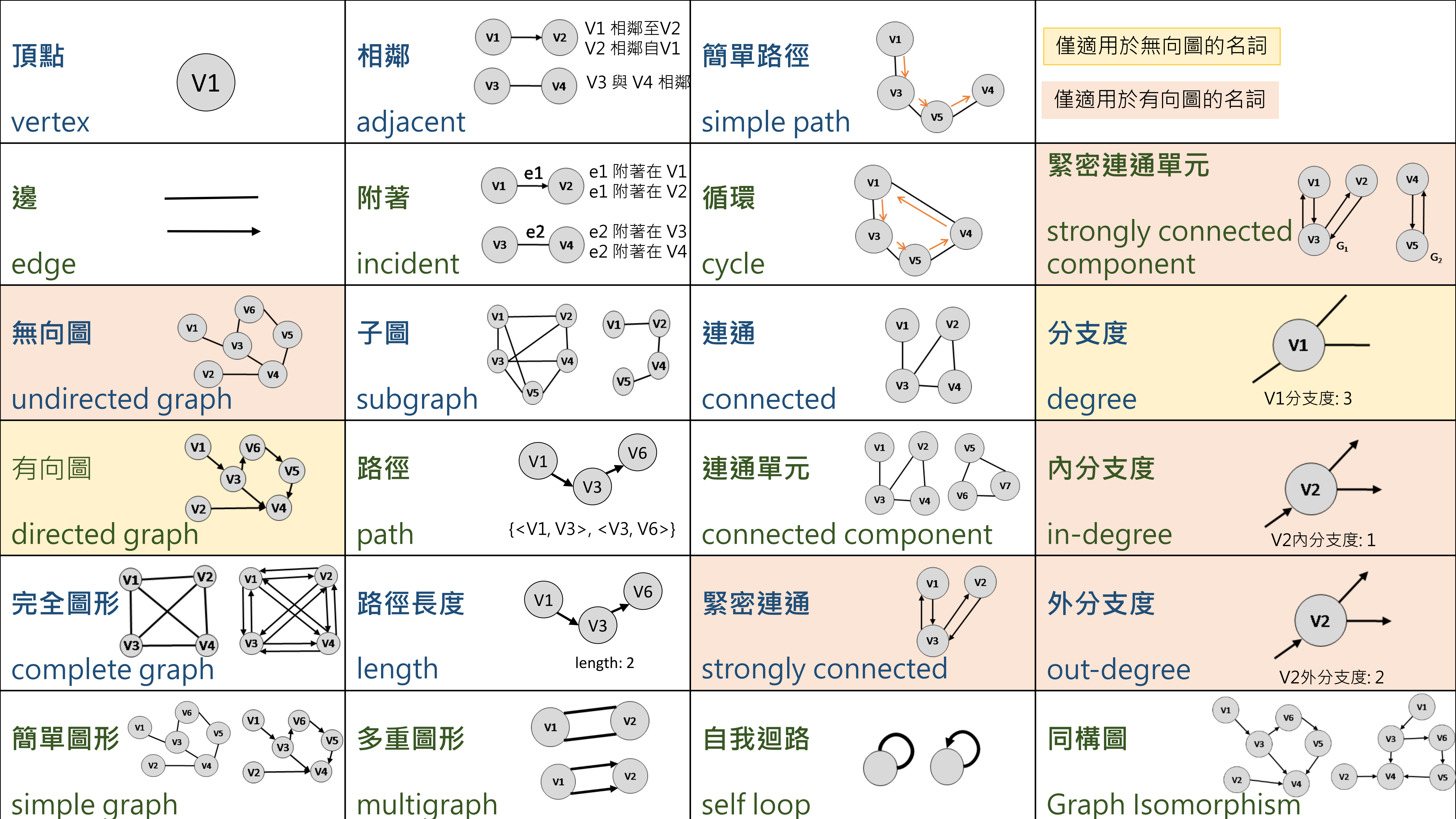


無向圖內的多重邊



概念2: 名詞整理

頂點 vertex	相鄰 adjacent V1 相鄰至V2 V2 相鄰自V1 V3 與 V4 相鄰	簡單路徑 simple path	僅適用於無向圖的名詞 僅適用於有向圖的名詞
邊 edge	附著 incident e1 附著在 V1 e1 附著在 V2 e2 附著在 V3 e2 附著在 V4	循環 cycle	緊密連通單元 strongly connected component
無向圖 undirected graph	子圖 subgraph	連通 connected	分支度 degree V1分支度: 3
有向圖 directed graph	路徑 path {<V1, V3>, <V3, V6>}	連通單元 connected component	內分支度 in-degree V2內分支度: 1
完全圖形 complete graph	路徑長度 length length: 2	緊密連通 strongly connected	外分支度 out-degree V2外分支度: 2
簡單圖形 simple graph	多重圖形 multigraph	自我迴路 self loop	同構圖 Graph Isomorphism



頂點
vertex
V1
相鄰
adjacent
V1 相鄰至V2
V2 相鄰自V1
V3 與 V4 相鄰
簡單路徑
simple path
僅適用於無向圖的名詞
僅適用於有向圖的名詞
邊
edge
附著
incident
e1
e2
e1 附著在 V1
e1 附著在 V2
e2 附著在 V3
e2 附著在 V4
循環
cycle
緊密連通單元
strongly connected component
G_1
G_2
無向圖
undirected graph
子圖
subgraph
連通
connected
分支度
degree
V1
V1分支度: 3
有向圖
directed graph
路徑
path
{<V1, V3>, <V3, V6>}
連通單元
connected component
內分支度
in-degree
V2
V2內分支度: 1
完全圖形
complete graph
路徑長度
length
length: 2
緊密連通
strongly connected
外分支度
out-degree
V2
V2外分支度: 2
簡單圖形
simple graph
多重圖形
multigraph
自我迴路
self loop
同構圖
Graph Isomorphism