C/C++ 進階班
演算法

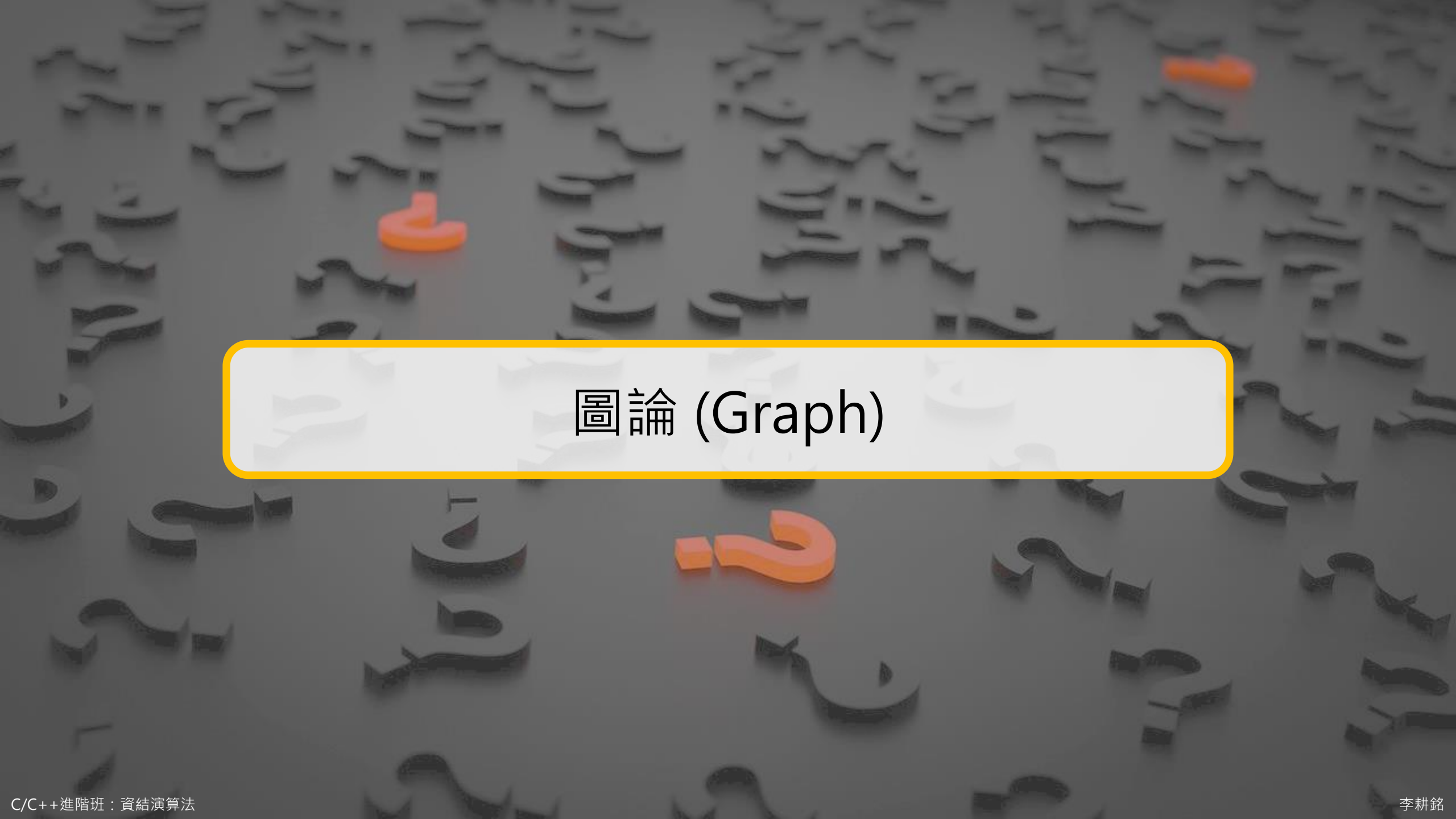
圖論
(Graph)

李耕銘

課程大綱

- 圖論 (Graph)
- 圖的表達方式
- 圖的分類
- 實戰練習

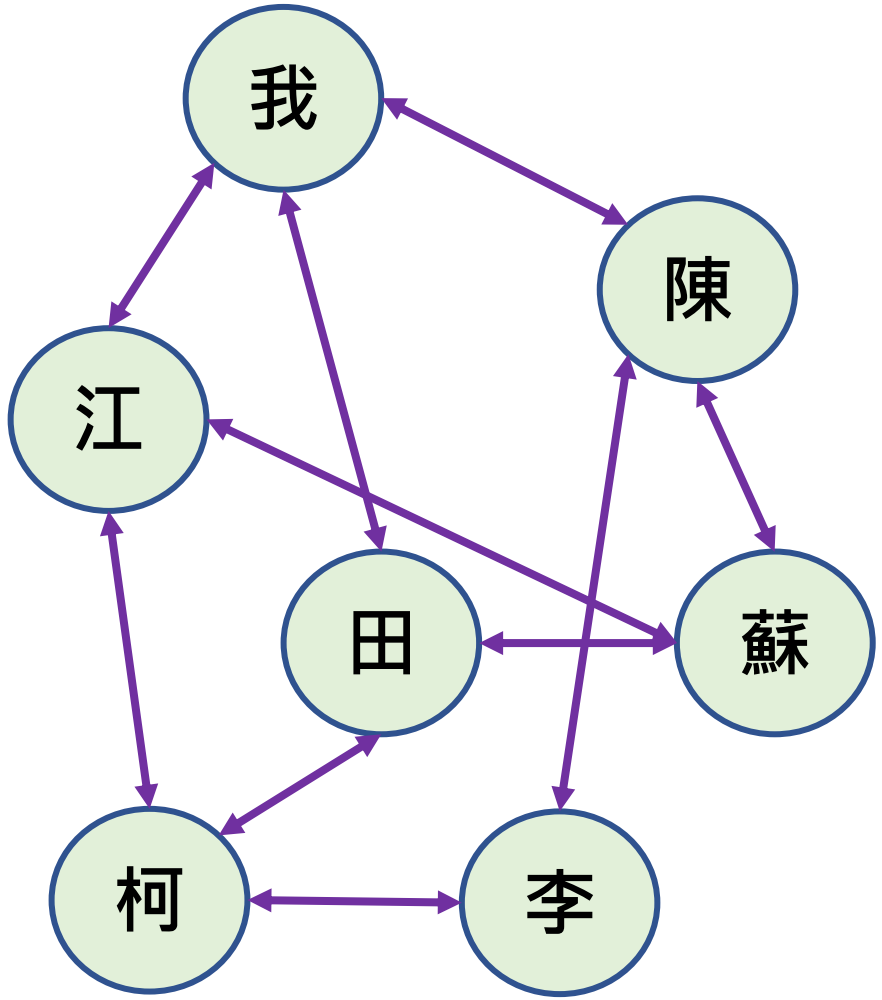
本章對於圖的分類&定義可以當參考



圖論 (Graph)

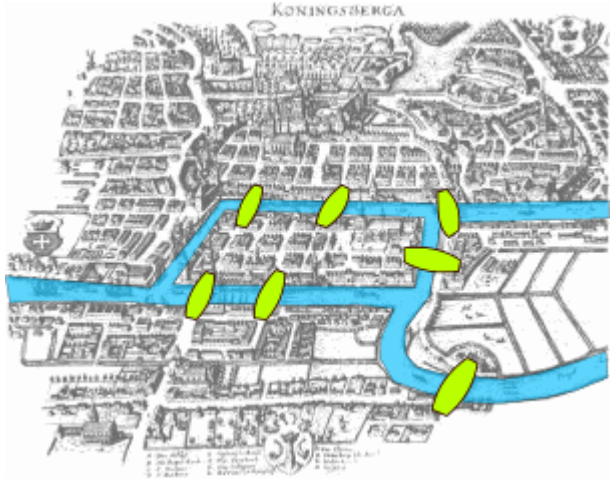
圖論 (Graph)

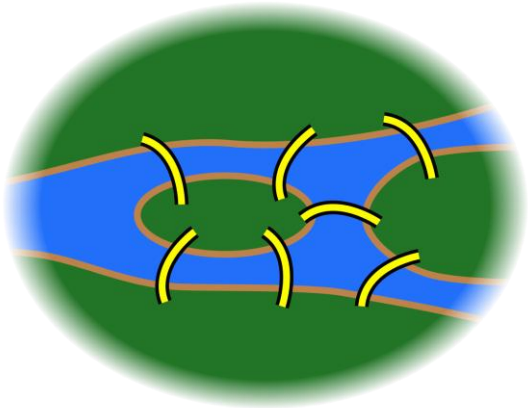
- 假設全班有多位同學
- 我的通訊錄只有一部分熟識的好朋友
- 但我記得所有同學的通訊錄有誰
- 假設我要通知訊息給小蘇
  - ➢ 透過同學一步步傳遞訊息
    - ✓ 我→小陳→小蘇
    - ✓ 我→小田→小蘇
- 要如何生成中間的路徑？

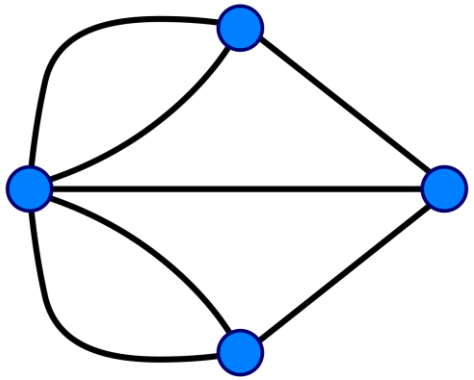


圖論 (Graph)

- (1753) 柯尼斯堡七橋問題（Seven Bridges of Königsberg）
  - 河中有兩個小島
  - 小島與河的兩岸有七條橋連接
- 所有橋都只能走一遍的前提下，如何才能把這個地方所有的橋都走遍？



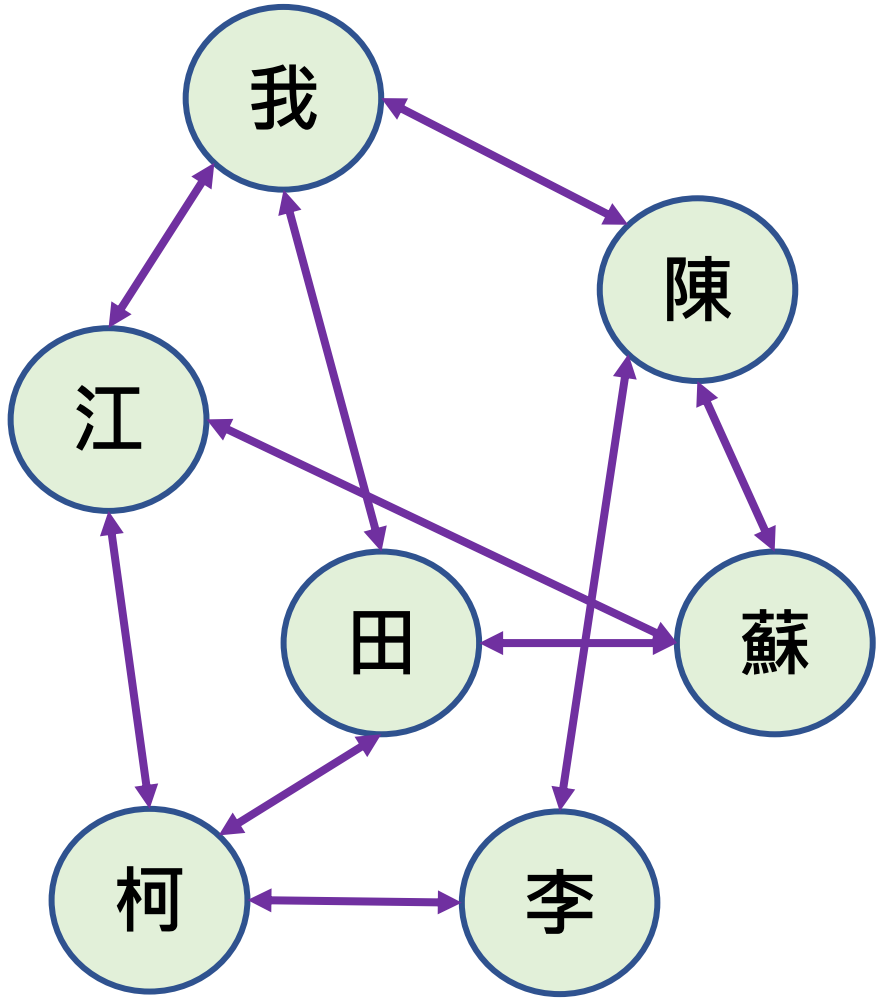




Wikipedia

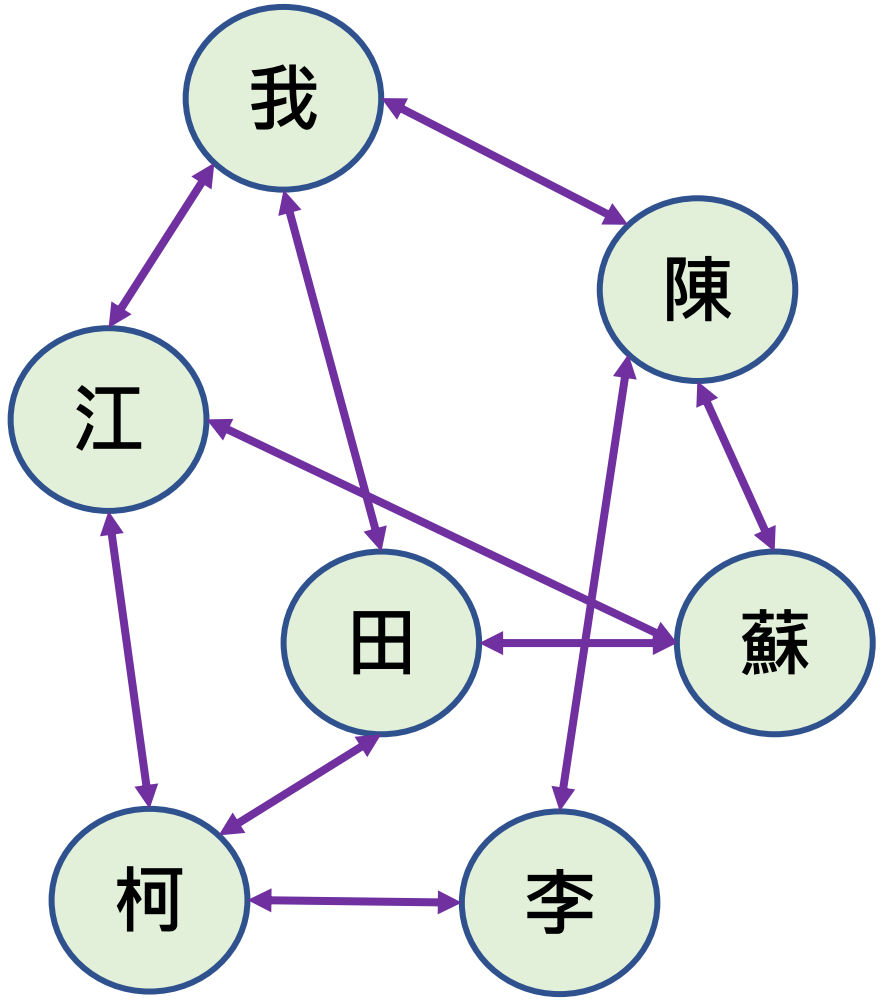
圖論的應用

- 表達&處理資料與資料間的關係
  - 地圖關係
  - 交通時間
  - 連通與否
  - 人際網路
  - 轉發封包



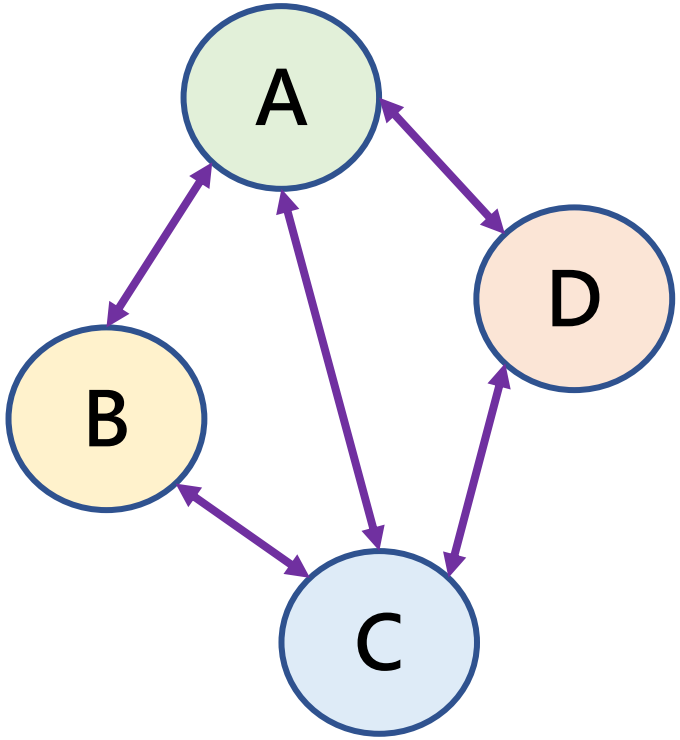
圖論的定義

- 由兩種資料構成
  1. 頂點(vertex)或節點 (nodes)
  2. 邊(edge)
- 數學的表示法：G = (V , E)
  - V(G)：G 的頂點集合
    - ✓ |V(G)|：頂點個數
  - E(G)：G 的邊集合
    - ✓ $e(v_1, v_2)$：連接v_1, v_2的邊
    - ✓ | E(G) |：邊的個數



圖論的定義

- G = (V , E)
  - V(G) = {A, B, C, D}
    - ✓ |V(G)| = 4
  - E(G) = {(A,B), (A,C), (A,D), (B,C), (C,D),}
    - ✓ | E(G) | = 5



圖論的定義

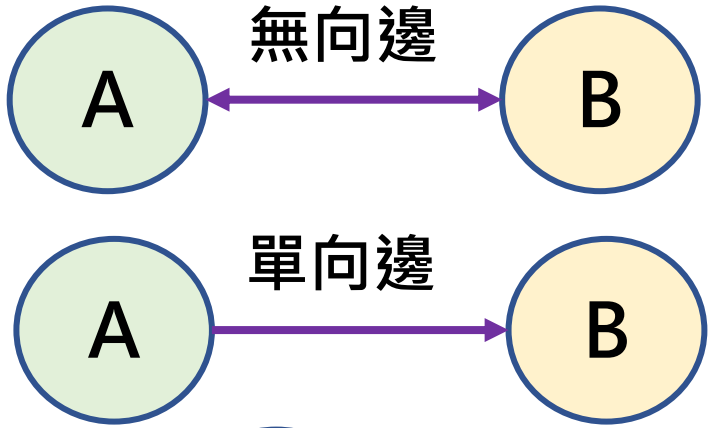
- 邊的種類 (同樣的邊可以有很多條)
  1. 方向
     - A. 雙向邊、無向邊 (undirected)

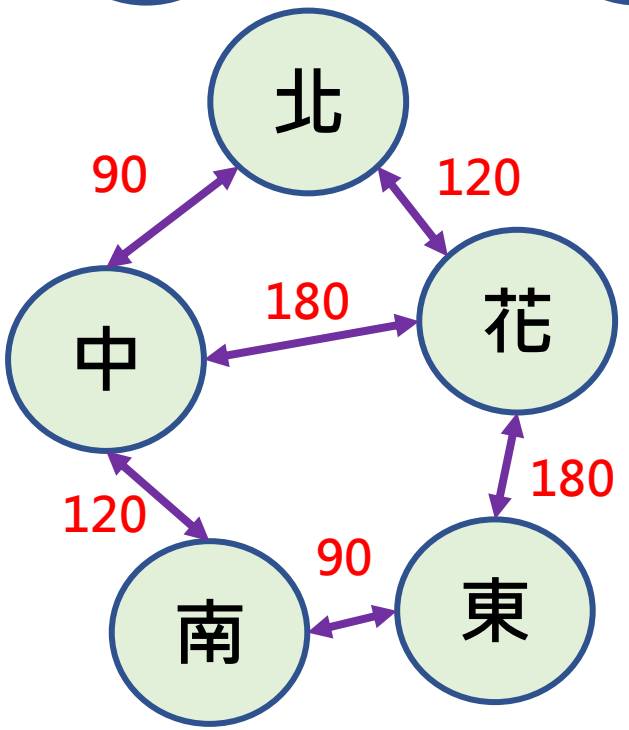
       $e(v_1, v_2) = e(v_2, v_1)$
     - B. 單向邊 (directed)

       $e(v_1, v_2) \neq e(v_2, v_1)$
  2. 權重
     - A. 有權重 (weighted)

       距離、時間、成本
     - A. 無權重 (unweighted)

       可視為所有邊的權重一致

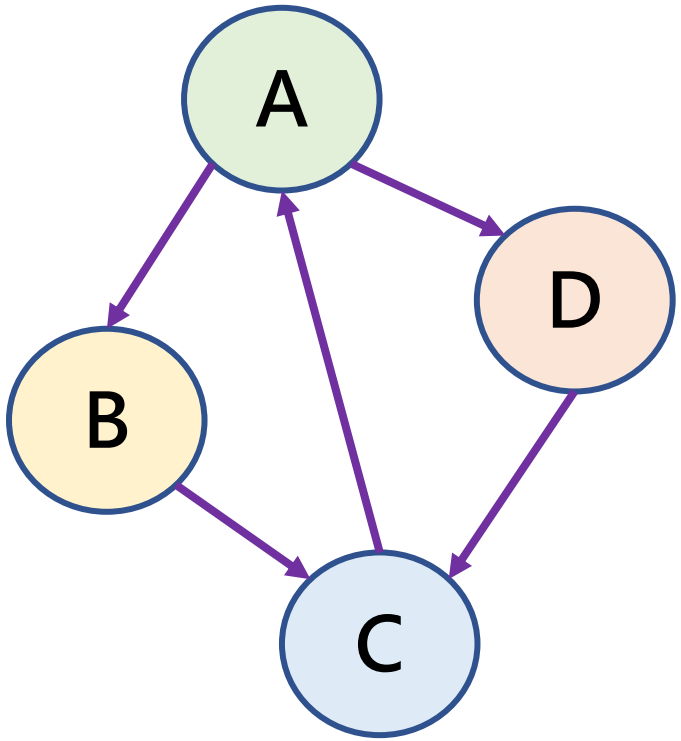




有權重

圖論的定義

- 度數 (Degree)
  - 該頂點連接的邊數目
  - Ex： $deg(A) = 3$
- 有向圖的入度 (In-Degree)
  - 指向該頂點的有向邊數量
  - Ex： $deg^{-}(A) = 1$
- 有向圖的出度 (Out-Degree)
  - 從該點指向其餘節點的有向邊數量
  - Ex： $deg^{+}(A) = 2$

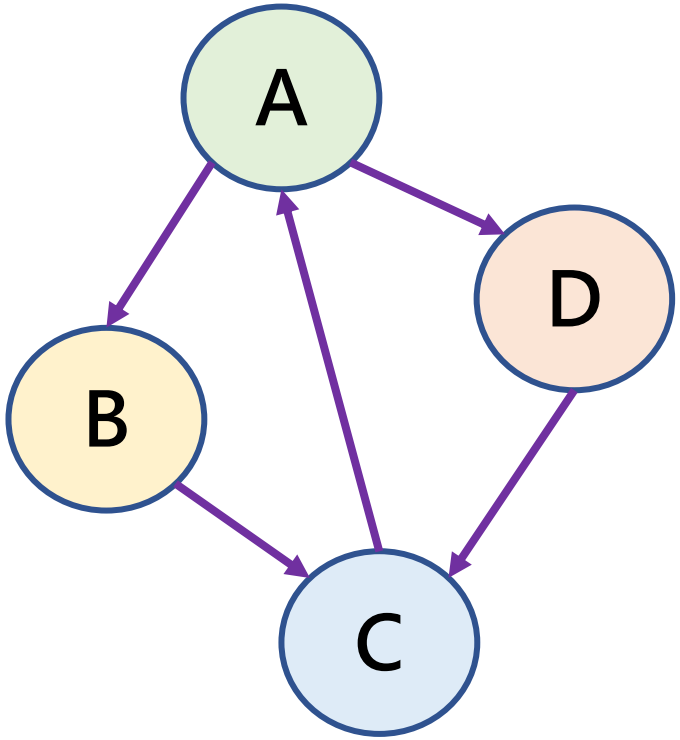


$$\sum_{i=1}^{N} \deg(E_i) = 2|E|$$

$$\sum_{i=1}^{N} \deg^{+}(E_i) + \sum_{i=1}^{N} \deg^{-}(E_i) = 2|E|$$

圖論的定義

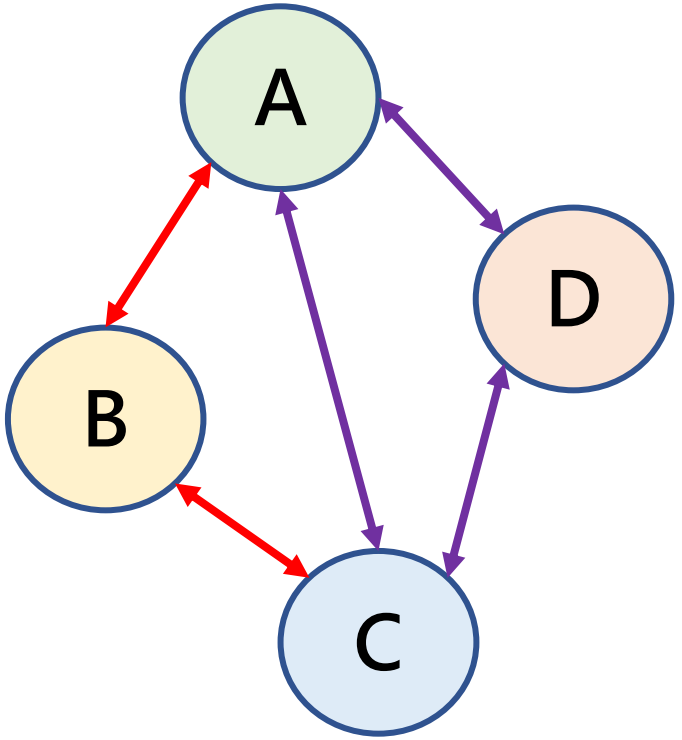
- 相鄰 (Adjacent)
  - v_1, v_2相鄰 *iff* $e(v_1, v_2)$ 或$e(v_2, v_1)$存在
  - 若 $e(v_1, v_2)$ 存在，則 v_1, v_2 相鄰
    - ✓ $e(v_1, v_2)$ $incident\ to\ v_1\ and\ v_2$
  - 有向圖 (Directed Graph)
    - ✓ v_1 $is\ adjacent\ to\ v_2$
    - ✓ v_2 $is\ adjacent\ from\ v_1$
  - Ex：*A is adjacent to B and D.*



$incident$：附著

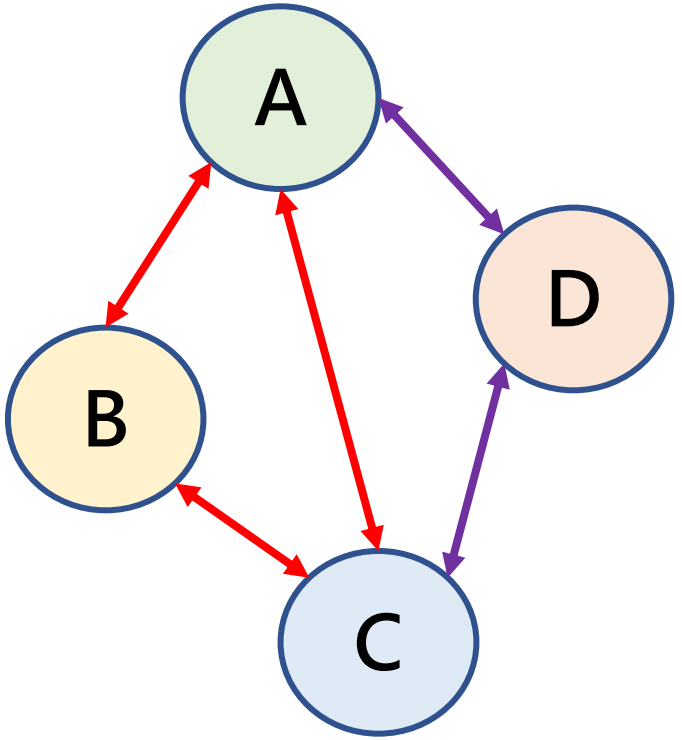
圖論的定義

- 路徑 (Path)
  - $v_1, e_1, v_2, e_2, v_3 \dots\dots, e_{n-1}, v_n$
  - $v \in V(G); e \in E(G); e_i = e(v_i, v_{i+1})$
  - Ex：A, (A,B), B, (B,C), C
- 簡單路徑 (Simple Path)
  - 每個點與邊只會使用一次，點跟邊不重複



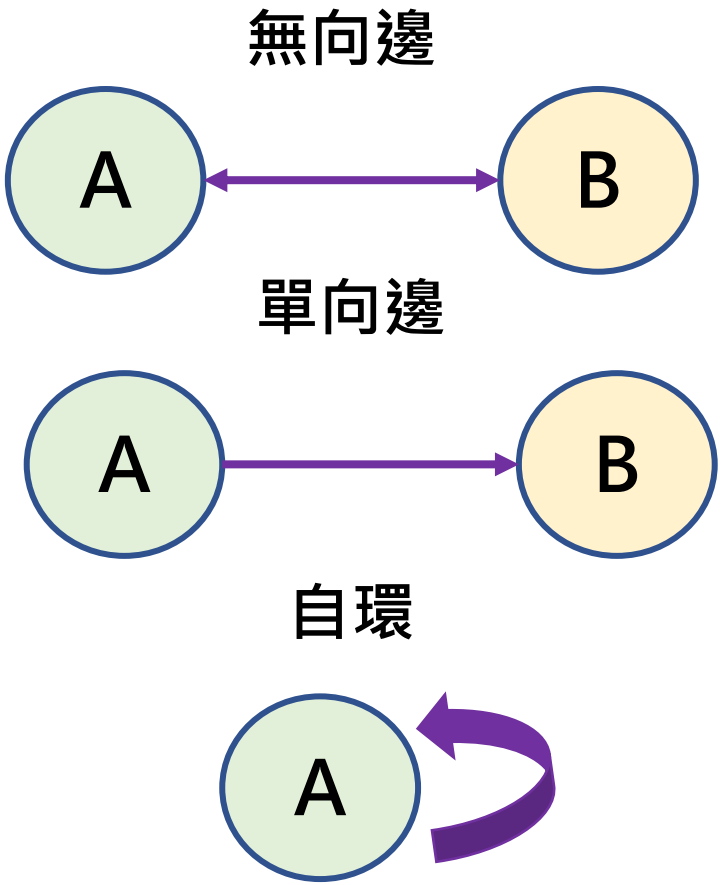
圖論的定義

- 環 (Cycle)
  - $v_1, e_1, v_2, e_2, v_3 \dots\dots, e_{n-1}, v_n$
  - 一條路徑中的$v_1 = v_n$，且至少有一條邊
  - Ex：A, (A,B), B, (B,C), C, (C,A), A
- 簡單環 (Simple Cycle)
  - 環的路徑中，除起點與終點外都不重複



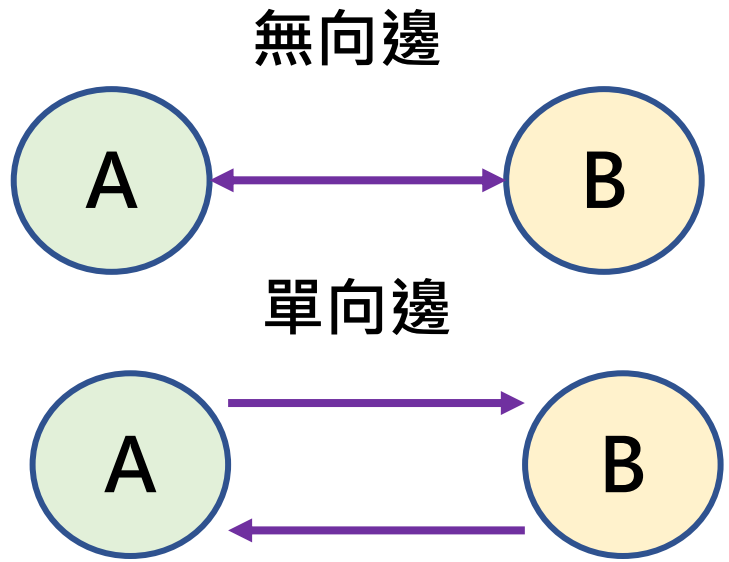
圖論的定義

- 無向圖 (Undirected Graph)
  - 所有的邊皆為無向邊
  - 不允許自環 (self-loop)
- 有向圖 (Directed Graph)
  - 所有的邊皆為有向邊
  - 允許自環 (self-loop)
- 混和圖 (Mixed Graph)
  - 有向邊、無向邊都有



圖論的定義

- 無向邊轉成有向邊
  - 生成兩條對向的有向邊
  - $e(v_1, v_2)$、$e(v_2, v_1)$
- 無向圖轉成有向圖
  - 以所有無向邊生成兩條對向、同權重的有向邊
- 有向圖轉成無向圖
  - 無法直接轉
- 解決有向圖就能夠解決無向圖！



Practice

Mission

Try LeetCode #997. Find the Town Judge

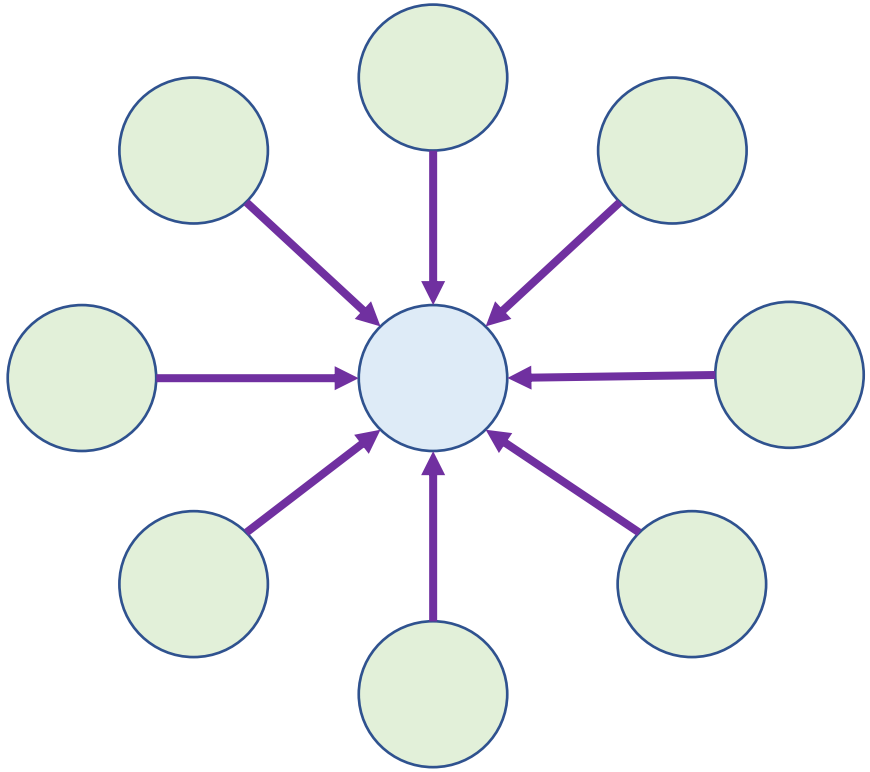
In a town, there are n people labelled from 1 to n. There is a rumor that one of these people is secretly the town judge. If the town judge exists, then:

1. The town judge trusts nobody.
2. Everybody (except for the town judge) trusts the town judge.
3. There is exactly one person that satisfies properties 1 and 2.

You are given trust, an array of pairs trust[i] = [a, b] representing that the person labelled a trusts the person labelled b. If the town judge exists and can be identified, return the label of the town judge. Otherwise, return -1.

Ref：https://leetcode.com/problems/find-the-town-judge/

Practice



Practice

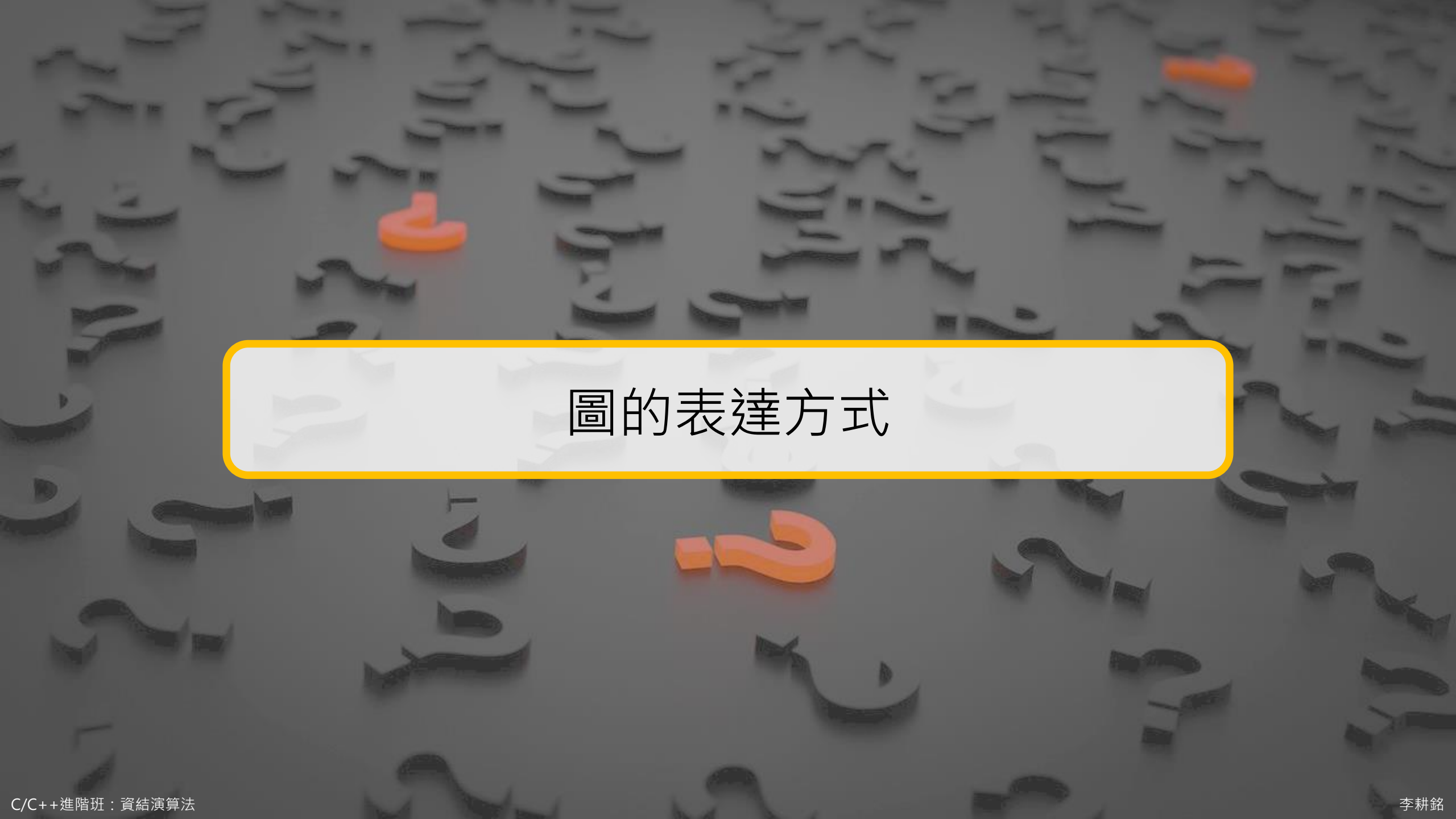
Mission

Try LeetCode #1791. Find Center of Star Graph

There is an undirected star graph consisting of n nodes labeled from 1 to n. A star graph is a graph where there is one center node and exactly n - 1 edges that connect the center node with every other node.

You are given a 2D integer array edges where each edges[i] = [ui, vi] indicates that there is an edge between the nodes ui and vi. Return the center of the given star graph.

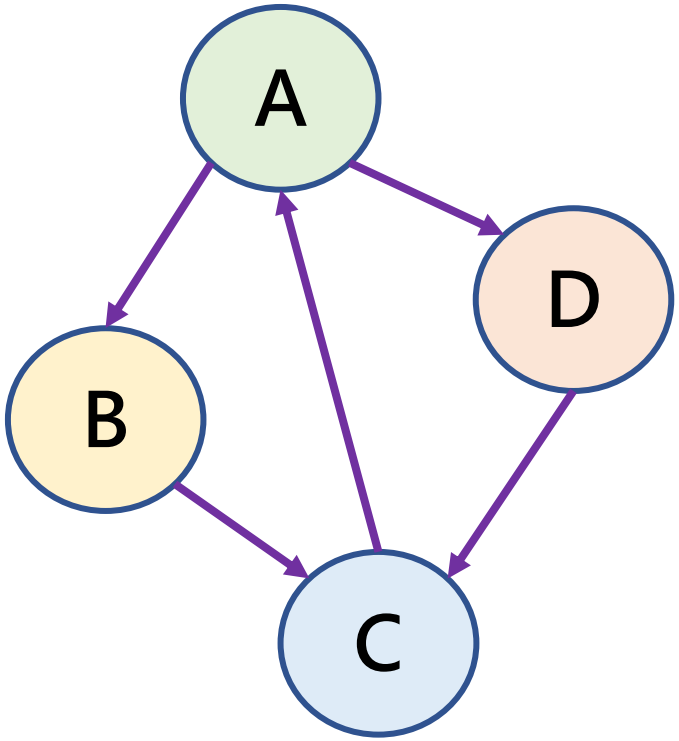
Ref：https://leetcode.com/problems/find-center-of-star-graph/



圖的表達方式

圖的表達方式

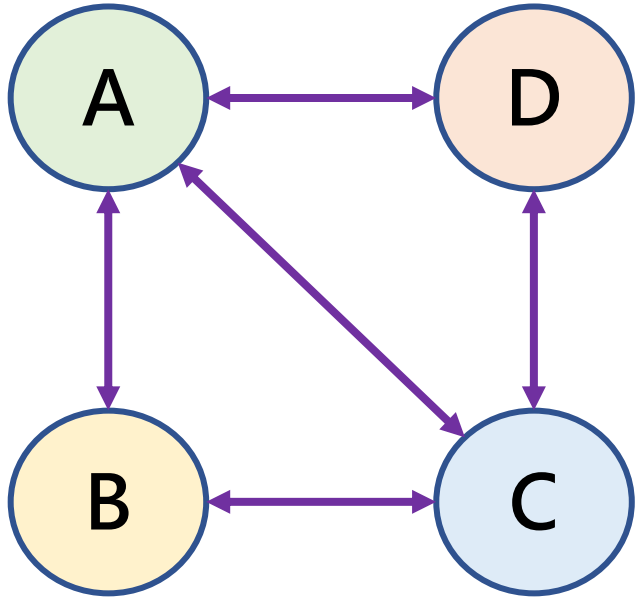
1. 鄰接矩陣 (Adjacent Matrix)
   - 利用矩陣來儲存圖中的頂點連接情形
2. 鄰接列表 (Adjacent List)
   - 各頂點都有個陣列 / 鏈結串列紀錄該點上的所有邊
3. 前向星 (Forward Star)
   - 所有邊都存在同陣列裡，並加以排序



鄰接矩陣 (Adjacent Matrix)

1. 鄰接矩陣 (Adjacent Matrix)
   - 利用矩陣來儲存圖中的頂點連接情形
   - 實作容易
   - 無法處理重邊

1. 時間複雜度：$O(|V|^2)$
2. 空間複雜度：$O(|V|^2)$
3. 查詢複雜度：$O(1)$
4. 修改複雜度：$O(1)$



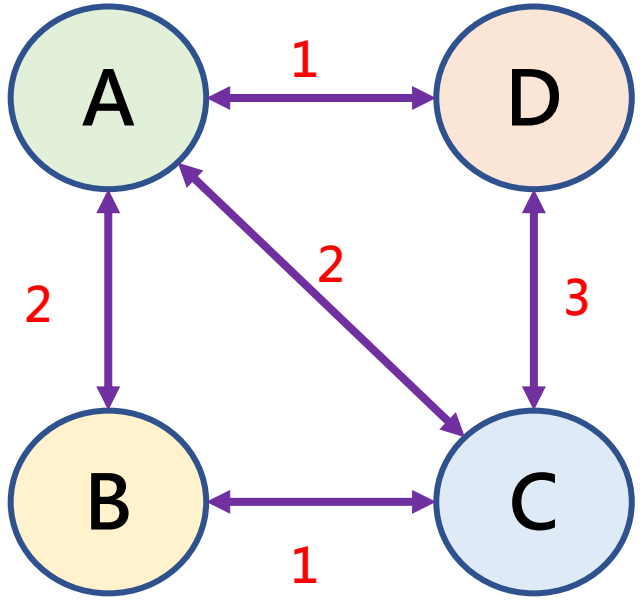
	A	B	C	D
A	0	1	1	1
B	1	0	1	0
C	1	1	0	1
D	1	0	1	0

對角對稱，無向圖可以只存一半

鄰接矩陣 (Adjacent Matrix)

1. 鄰接矩陣 (Adjacent Matrix)
   - 利用矩陣來儲存圖中的頂點連接情形
   - 實作容易
   - 無法處理重邊

1. 時間複雜度：$O(|V|^2)$
2. 空間複雜度：$O(|V|^2)$
3. 查詢複雜度：$O(1)$
4. 修改複雜度：$O(1)$



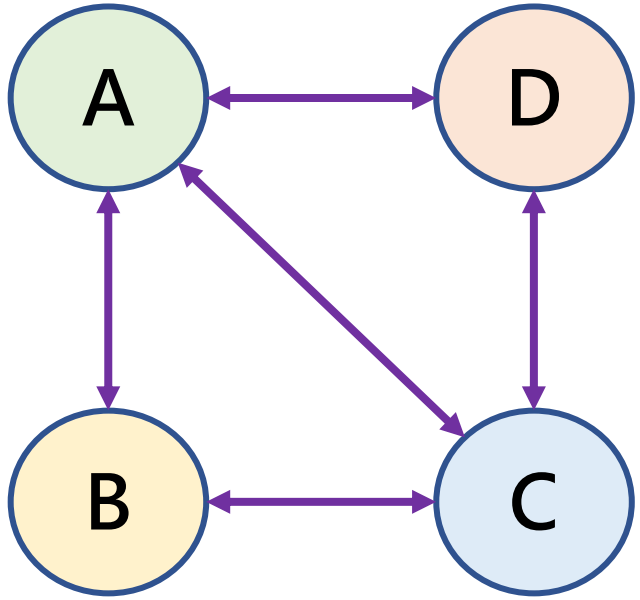
	A	B	C	D
A	0	2	2	1
B	2	0	1	0
C	2	1	0	3
D	1	0	3	0

對角對稱，無向圖可以只存一半

鄰接列表 (Adjacent List)

2. 鄰接列表 (Adjacent List)
   - ➢ 陣列 / 鏈結串列紀錄該點上的所有邊
   - ➢ 連到同一點上的邊沒有特別順序
   - ➢ 可以處理重邊

1. 時間複雜度：$O(|V| + |E|)$
2. 空間複雜度：$O(|V| + |E|)$
3. 查詢複雜度：取決於資料結構
4. 修改複雜度：取決於資料結構

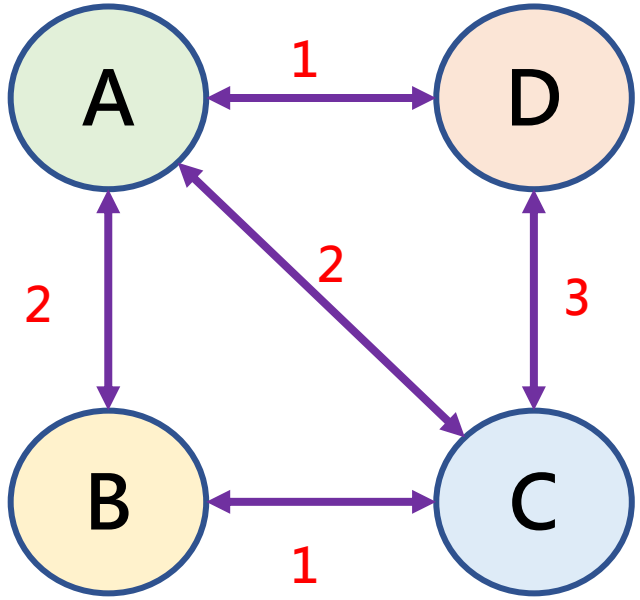


A	→B→C→D
B	→A→C
C	→B→A→D
D	→A→C

鄰接列表 (Adjacent List)

2. 鄰接列表 (Adjacent List)
   - ➢ 陣列 / 鏈結串列紀錄該點上的所有邊
   - ➢ 連到同一點上的邊沒有特別順序
   - ➢ 可以處理重邊

1. 時間複雜度：$O(|V| + |E|)$
2. 空間複雜度：$O(|V| + |E|)$
3. 查詢複雜度：取決於資料結構
4. 修改複雜度：取決於資料結構

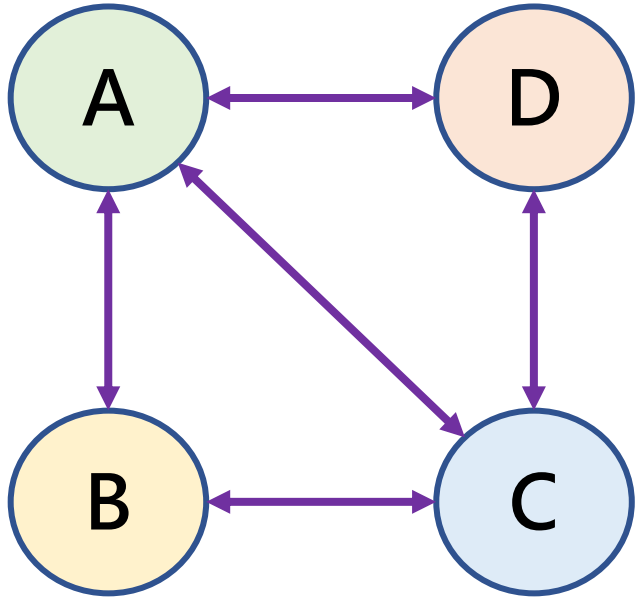


A	→B,2→C,2→D,1
B	→A,2→C,1
C	→B,1→A,2→D,3
D	→A,1→C,3

前向星 (Forward Star)

3. 前向星 (Forward Star)
   - ➢ 用一個陣列儲存所有邊
   - ➢ 紀錄每一個邊的起點與終點

1. 時間複雜度：$O(|V| + |E|)$
2. 空間複雜度：$O(|V| + |E|)$
3. 查詢複雜度：$O(|E|)$
4. 修改複雜度：$O(|E|)$

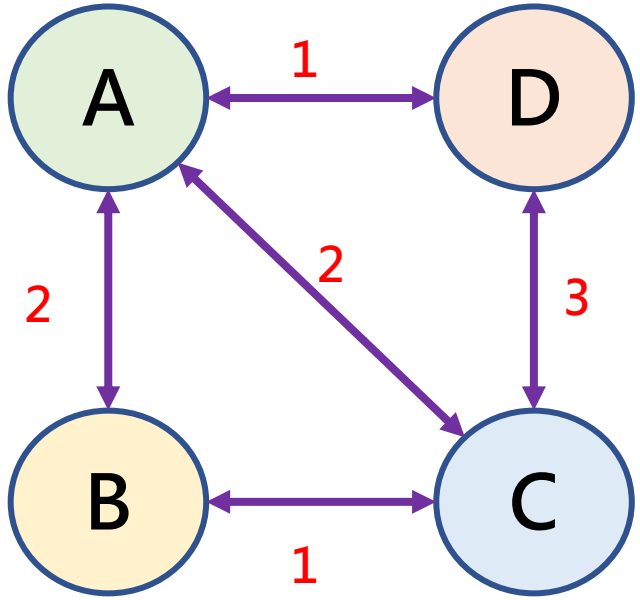


From	A	A	A	B	B	C	C	C	D	D
To	B	C	D	A	C	A	B	D	A	C

前向星 (Forward Star)

3. 前向星 (Forward Star)
   - ➢ 用一個陣列儲存所有邊
   - ➢ 紀錄每一個邊的起點與終點

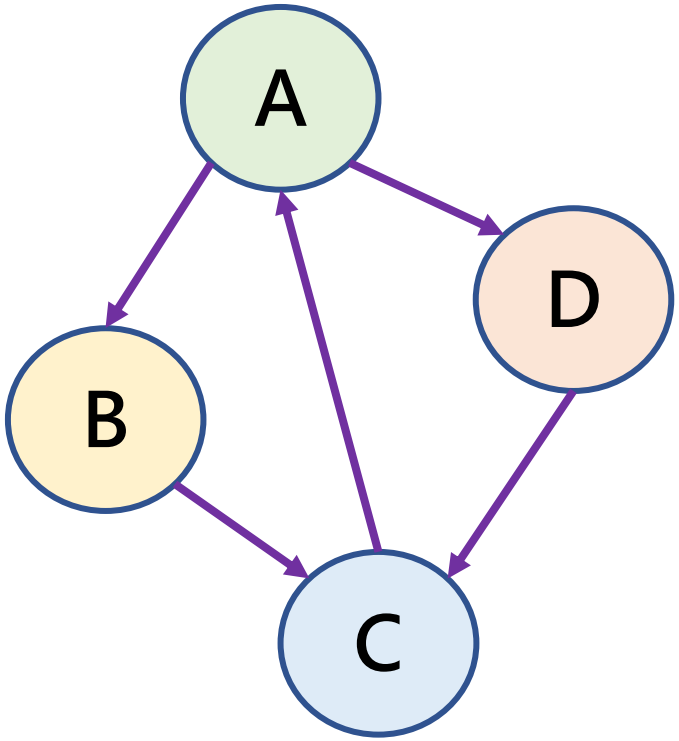
1. 時間複雜度：$O(|V| + |E|)$
2. 空間複雜度：$O(|V| + |E|)$
3. 查詢複雜度：$O(|E|)$
4. 修改複雜度：$O(|E|)$



From	A	A	A	B	B	C	C	C	D	D
Weight	2	2	1	2	1	2	1	3	1	3
To	B	C	D	A	C	A	B	D	A	C

圖的表達方式

1. 鄰接矩陣 (Adjacent Matrix)
   - 利用矩陣來儲存圖中的頂點連接情形
   - 優：查詢、新增快 $O(1)$
   - 缺：浪費空間 $O(|V|^2)$
   - 適用於稠密的圖 (dense)：$|E| \approx |V|^2$
2. 鄰接列表 (Adjacent List)
   - 各頂點都有個陣列 / 鏈結串列紀錄該點上的所有邊
   - 優：節省空間 $O(|V| + |E|)$
   - 缺：查詢、新增較花時間
   - 適用於稀疏的圖 (sparse)：$|E| \ll |V|^2$
3. 前向星 (Forward Star)
   - 所有邊都存在同陣列裡，並加以排序



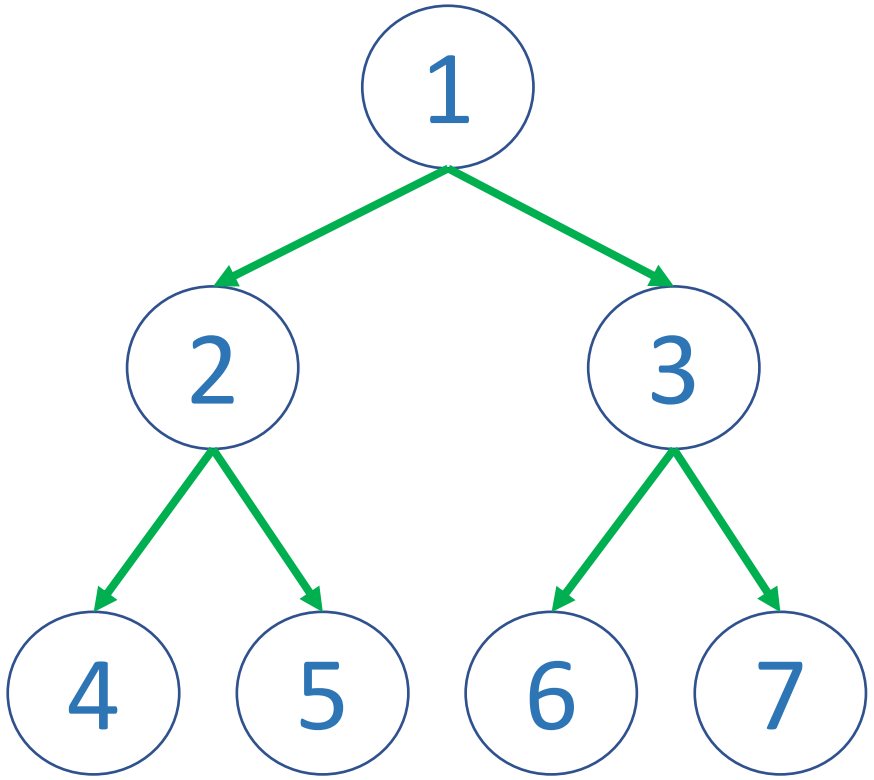
Example Code

Mission

試著用鄰接矩陣的方式儲存並處理圖，並且包含以下功能：

1. 新增一條邊
2. 印出所有邊

Practice

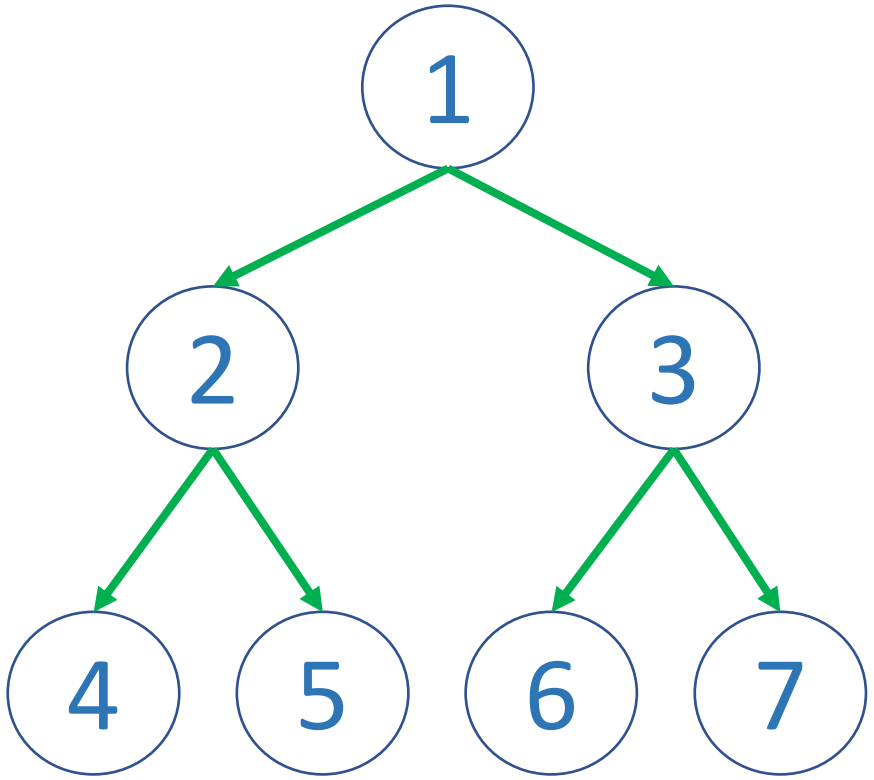


Mission

1. 為左邊的完滿二元樹建立鄰接矩陣的表示方式
2. 為左邊的完滿二元樹建立鄰接列表的表示方式

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. Introduction to Algorithms, Third Edition (3rd. ed.). The MIT Press.

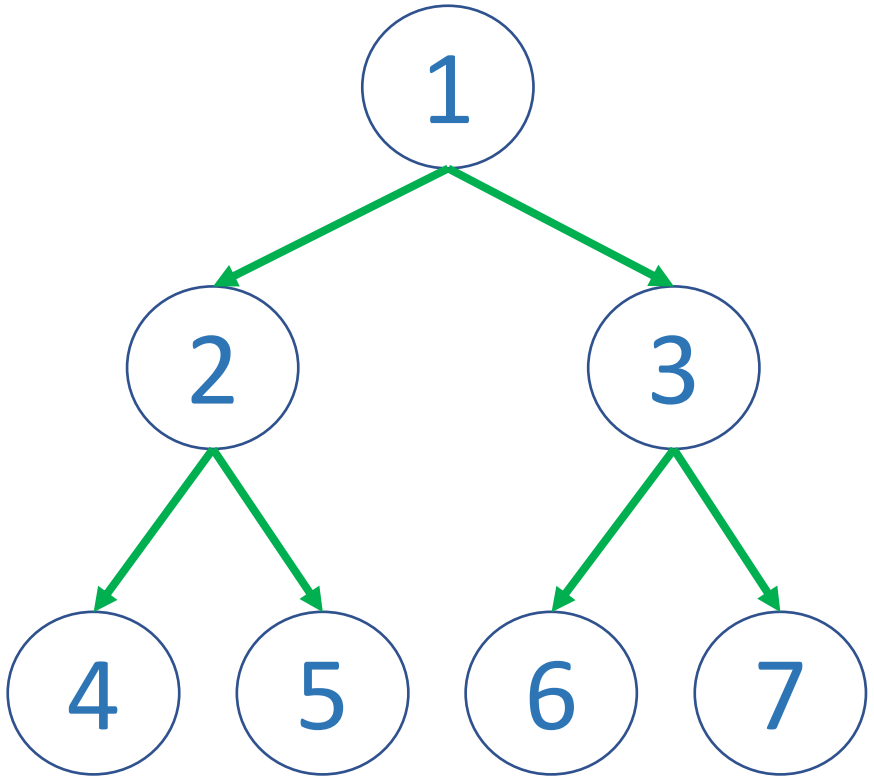
Practice



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

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. Introduction to Algorithms, Third Edition (3rd. ed.). The MIT Press.

Practice



1	→2→3
2	→4→5
3	→6→7
4	→
5	→
6	→
7	→

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. Introduction to Algorithms, Third Edition (3rd. ed.). The MIT Press.

Practice

Mission

試著用鄰接列表的方式儲存並處理圖，並且包含以下功能：

1. 新增一條邊
2. 印出所有邊

Practice

Mission

Try LeetCode #133. Clone Graph

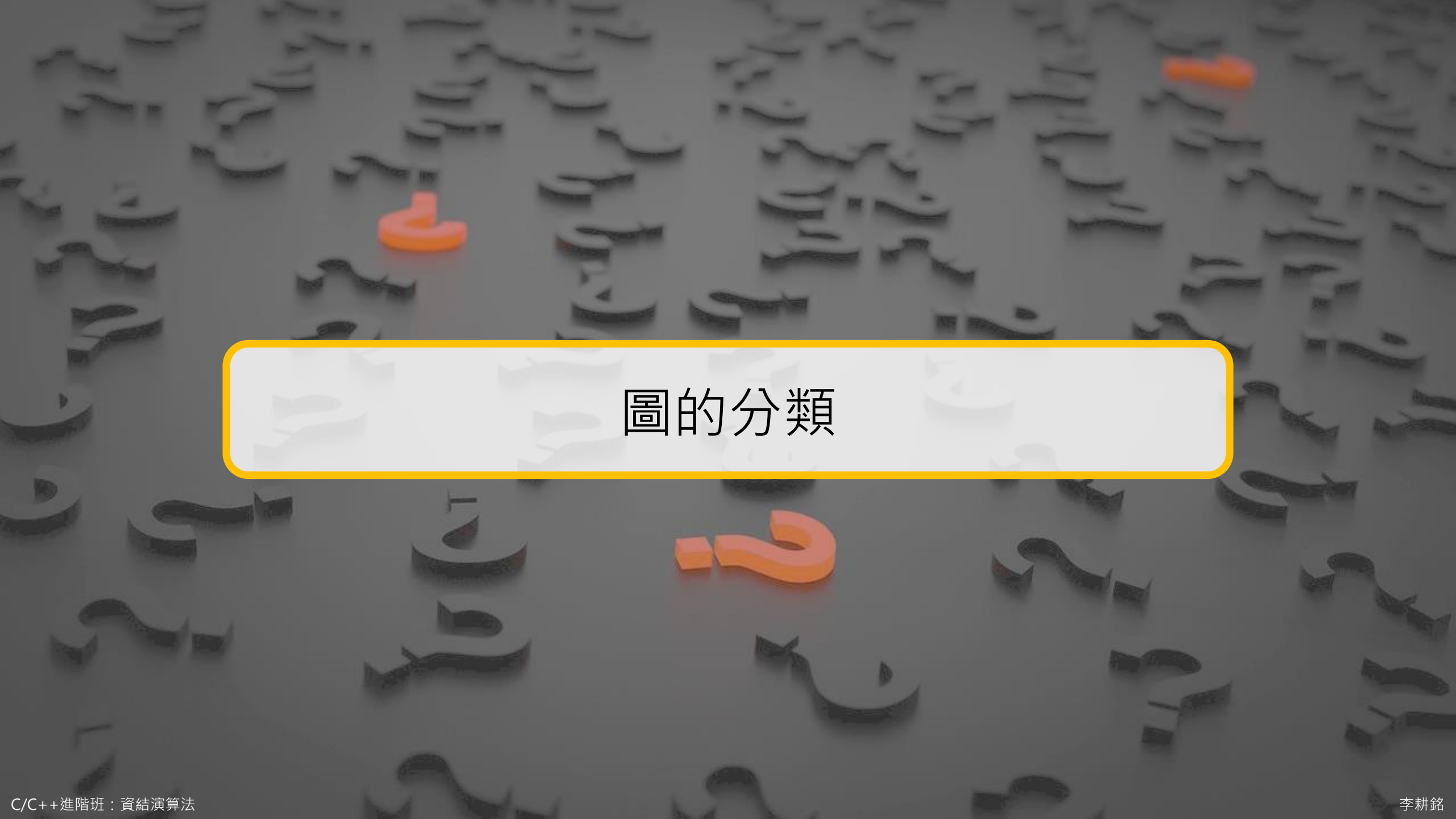
Given a reference of a node in a connected undirected graph.

Return a deep copy (clone) of the graph.

Each node in the graph contains a value (int) and a list (List[Node]) of its neighbors.

```
class Node {
    public int val;
    public List<Node> neighbors;
}
```

Ref：https://leetcode.com/problems/clone-graph/



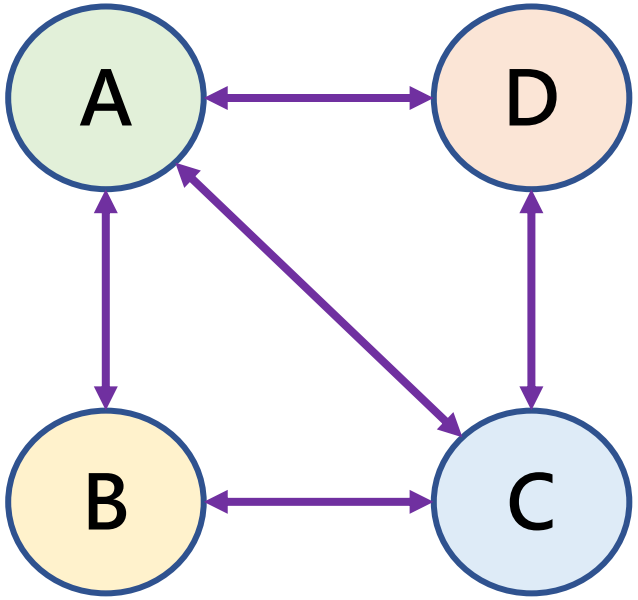
圖的分類

圖的分類

- 簡單圖 (Simple Graph)
- 有向無環圖 (DAG)
- AOV 網路 (Activity On Vertex)
- 拓樸排序 (Topological Sort)
- AOE 網路 (Activity On Edge)
- 二分圖 (Bipartite Graph)
- 平面圖 (Plannar Graph)
- 連通圖 (Connected Graph)
  - Strongly connected
  - Weekly connected
  - Disconnected
- 完全圖 (Complete Graph)
- Tree & Forest
- 子圖、補圖、生成樹

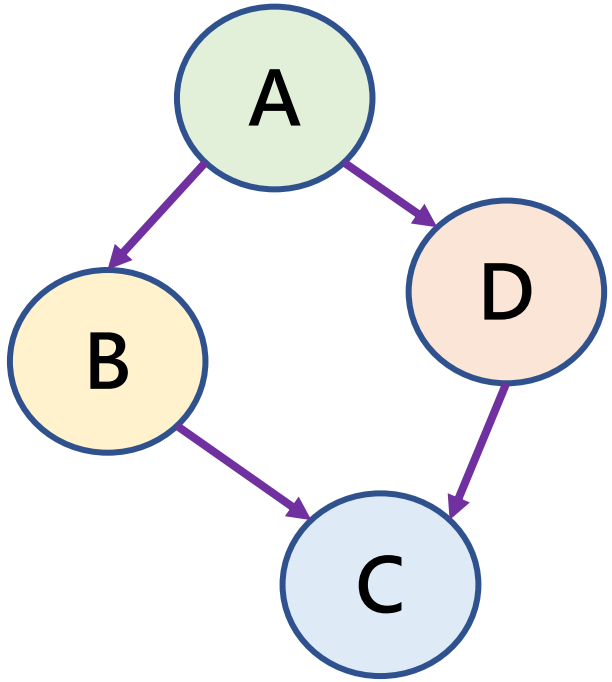
簡單圖 (Simple Graph)

- 簡單圖 (Simple Graph)
  - 不包含重邊 ($e_i = e_j, i \neq j$)
  - 不包含自環 ($e(v_i, v_i)$)



有向無環圖 (DAG)

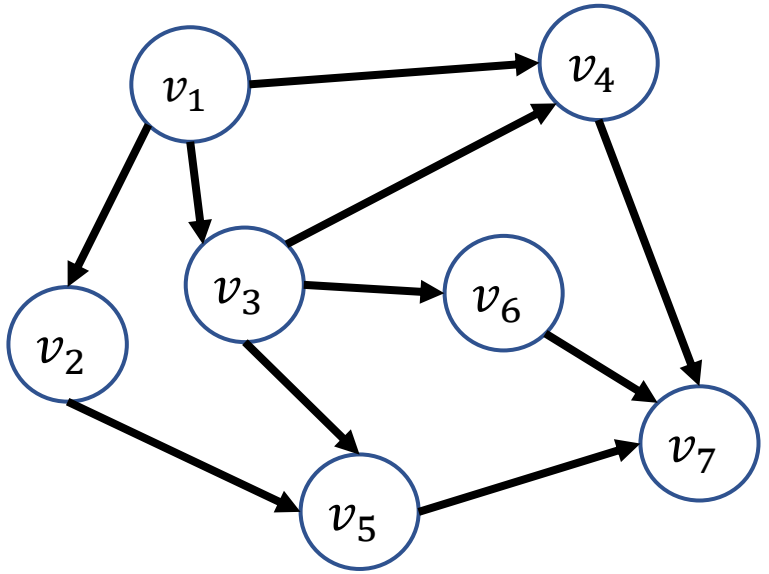
- 有向無環圖 (DAG)
  - Directed Acyclic Graph
  - 不具有環的有向圖
  - 動態規劃的過程是一張有向無環圖 (DAG)



AOV 網路

AOV 網路 (Activity On Vertex)

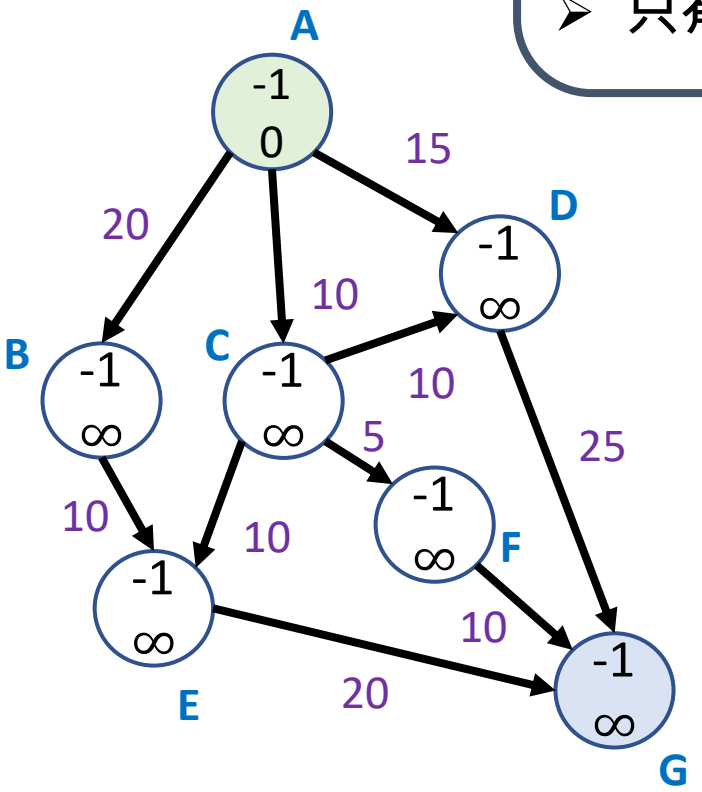
- 強調各活動的發生順序
  - ✓ 以頂點表示活動
  - ✓ 邊表示活動間的優先順序
    - $e(v_i, v_j)$ 表 v_i 的活動辦完後才能辦理 v_j

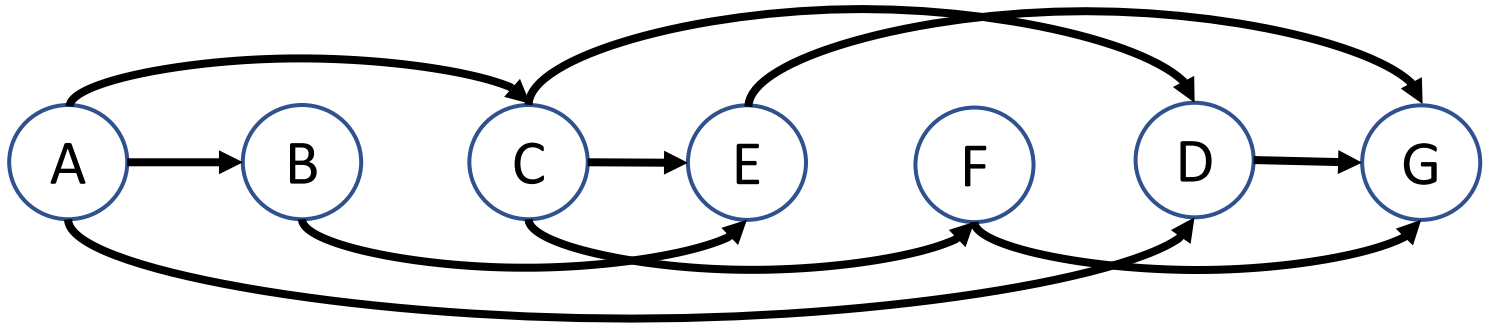


Topological Sort

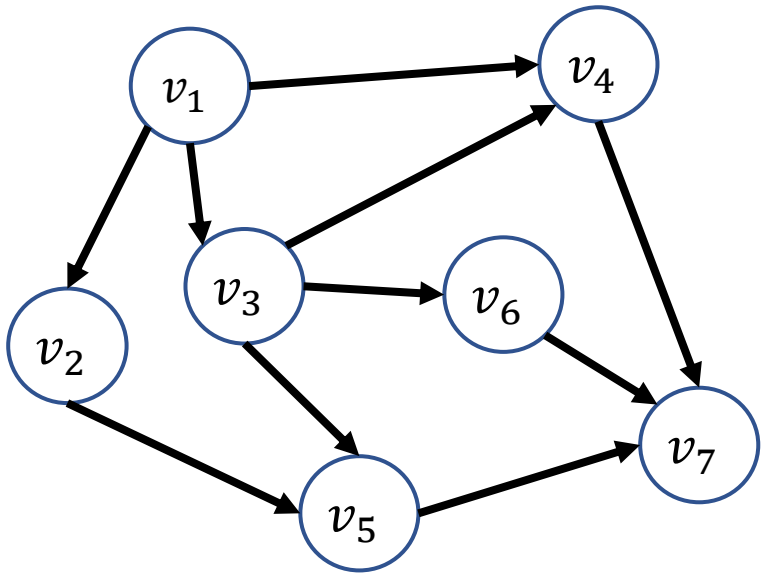
Topological Sort(拓撲排序)

- 每條在有向無環圖或 AOV 網路中的 Edge(A,B)
  - ✓ 拓樸排序必是 Vertex(A) 在 Vertex(B) 之前
- 只有有向無環圖的拓樸排序才有意義





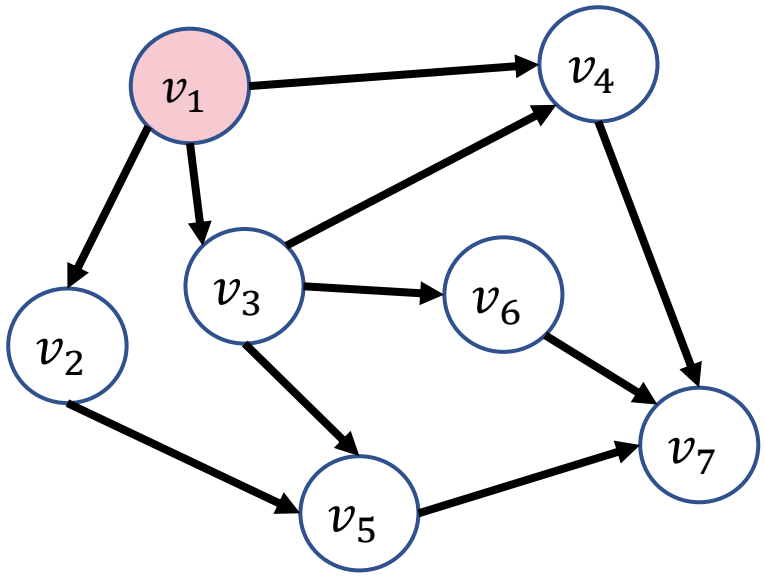
Topological Sort



AOV 網路中的 Topological Sort (拓撲排序)

1. 從其中一個入度為 0 的頂點出發並置入排序中
2. 把該頂點的所有鄰邊刪除
3. 重複步驟 1~2，直至找不到入度為 0 的點

Topological Sort

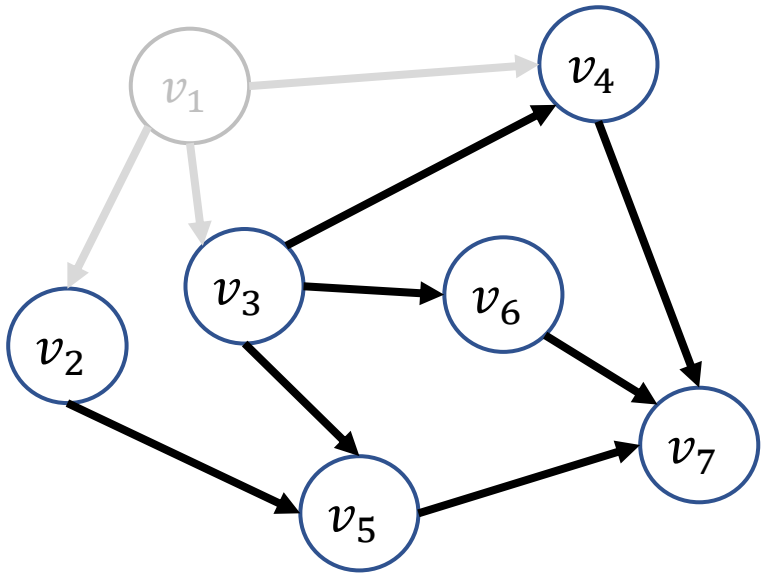


AOV 網路中的 Topological Sort (拓撲排序)

1. 從其中一個入度為 0 的頂點出發並置入排序中
2. 把該頂點的所有鄰邊刪除
3. 重複步驟 1~2，直至找不到入度為 0 的點

Topological Sort：v_1

Topological Sort

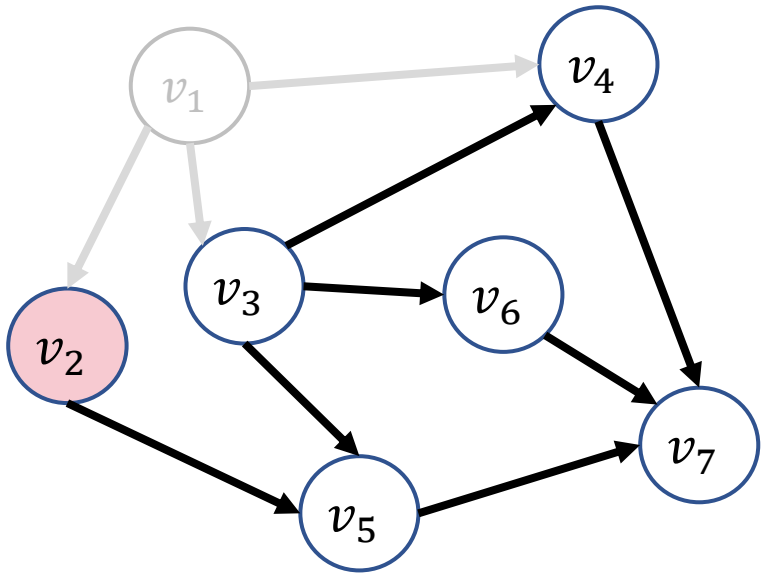


AOV 網路中的 Topological Sort (拓撲排序)

1. 從其中一個入度為 0 的頂點出發並置入排序中
2. 把該頂點的所有鄰邊刪除
3. 重複步驟 1~2，直至找不到入度為 0 的點

Topological Sort：v_1

Topological Sort

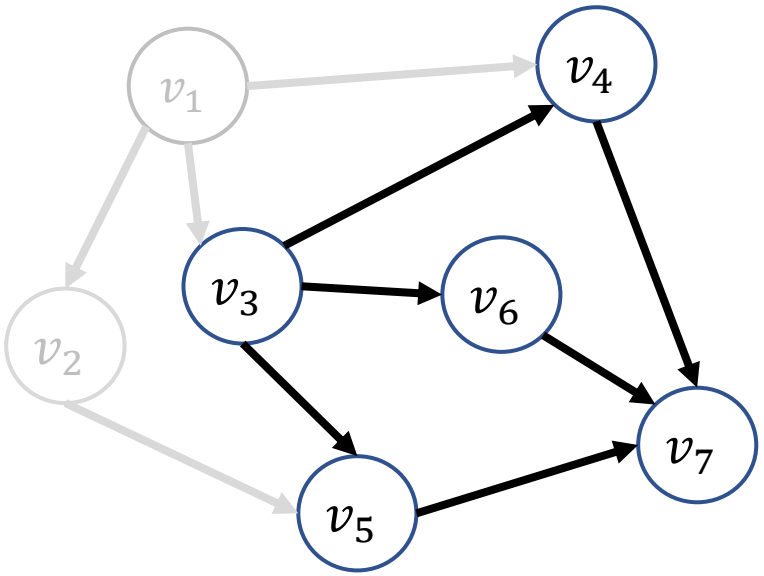


AOV 網路中的 Topological Sort (拓撲排序)

1. 從其中一個入度為 0 的頂點出發並置入排序中
2. 把該頂點的所有鄰邊刪除
3. 重複步驟 1~2，直至找不到入度為 0 的點

Topological Sort：v_1 v_2

Topological Sort

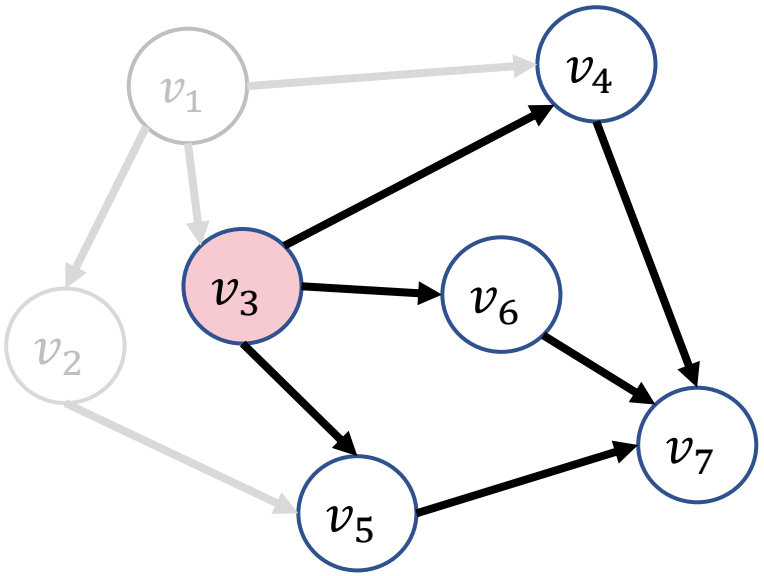


AOV 網路中的 Topological Sort (拓撲排序)

1. 從其中一個入度為 0 的頂點出發並置入排序中
2. 把該頂點的所有鄰邊刪除
3. 重複步驟 1~2，直至找不到入度為 0 的點

Topological Sort：v_1 v_2

Topological Sort

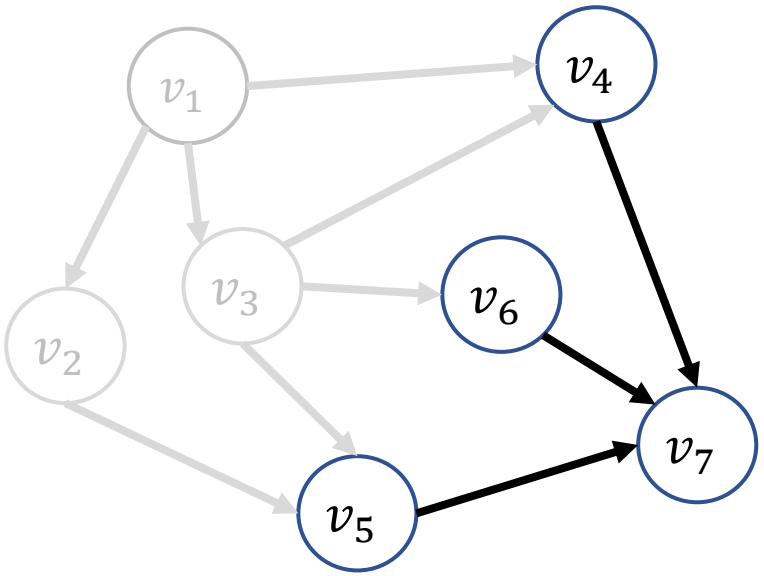


AOV 網路中的 Topological Sort (拓撲排序)

1. 從其中一個入度為 0 的頂點出發並置入排序中
2. 把該頂點的所有鄰邊刪除
3. 重複步驟 1~2，直至找不到入度為 0 的點

Topological Sort：v_1 v_2 v_3

Topological Sort

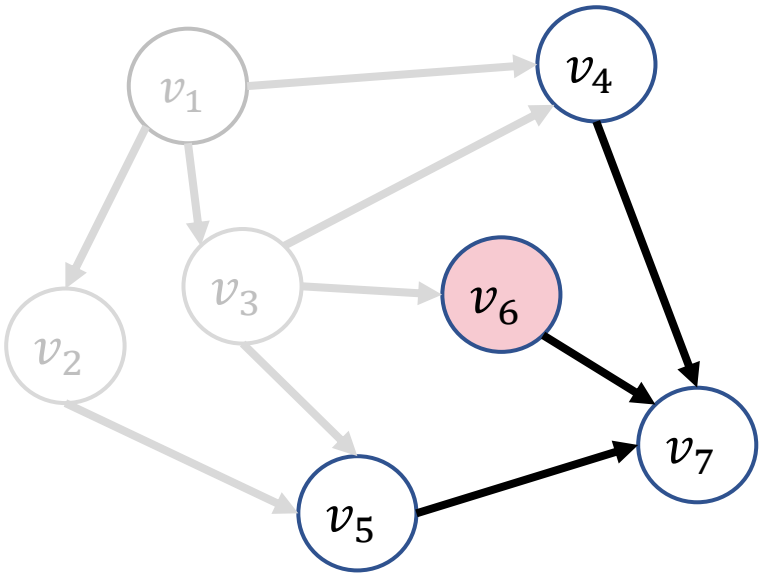


AOV 網路中的 Topological Sort (拓撲排序)

1. 從其中一個入度為 0 的頂點出發並置入排序中
2. 把該頂點的所有鄰邊刪除
3. 重複步驟 1~2，直至找不到入度為 0 的點

Topological Sort：v_1 v_2 v_3

Topological Sort

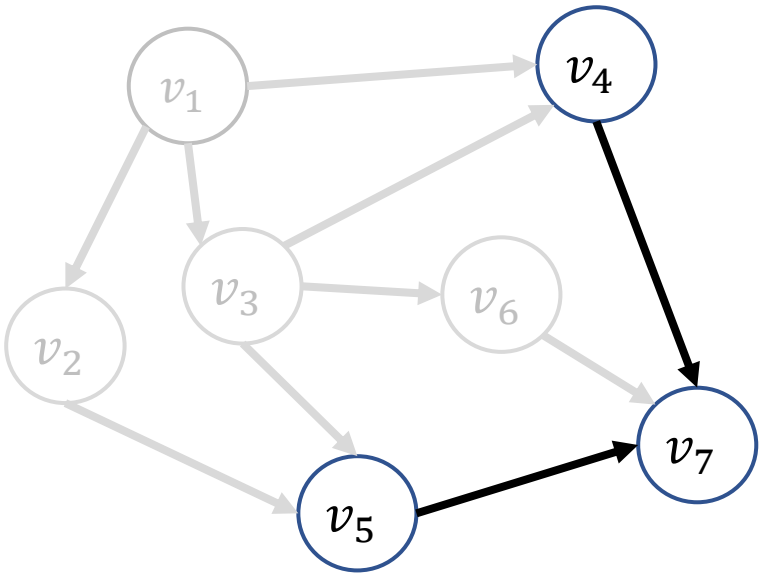


AOV 網路中的 Topological Sort (拓撲排序)

1. 從其中一個入度為 0 的頂點出發並置入排序中
2. 把該頂點的所有鄰邊刪除
3. 重複步驟 1~2，直至找不到入度為 0 的點

Topological Sort：v_1 v_2 v_3 v_6

Topological Sort

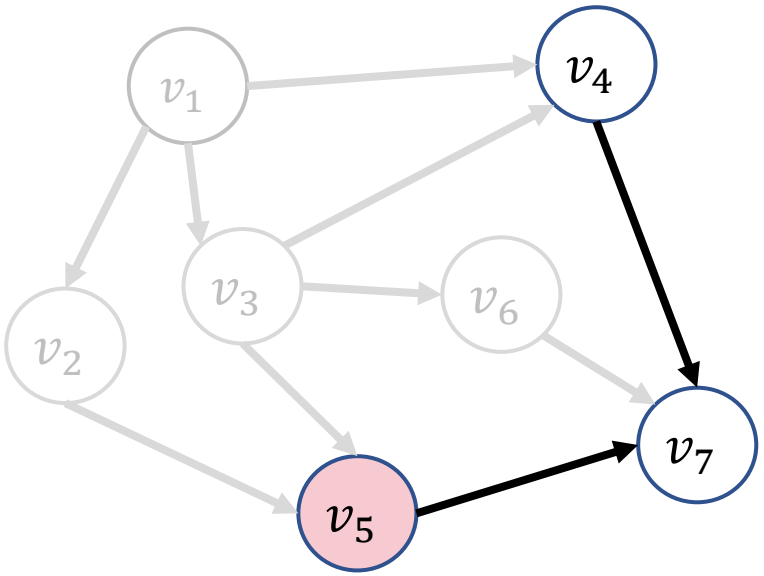


AOV 網路中的 Topological Sort (拓撲排序)

1. 從其中一個入度為 0 的頂點出發並置入排序中
2. 把該頂點的所有鄰邊刪除
3. 重複步驟 1~2，直至找不到入度為 0 的點

Topological Sort：v_1 v_2 v_3 v_6

Topological Sort

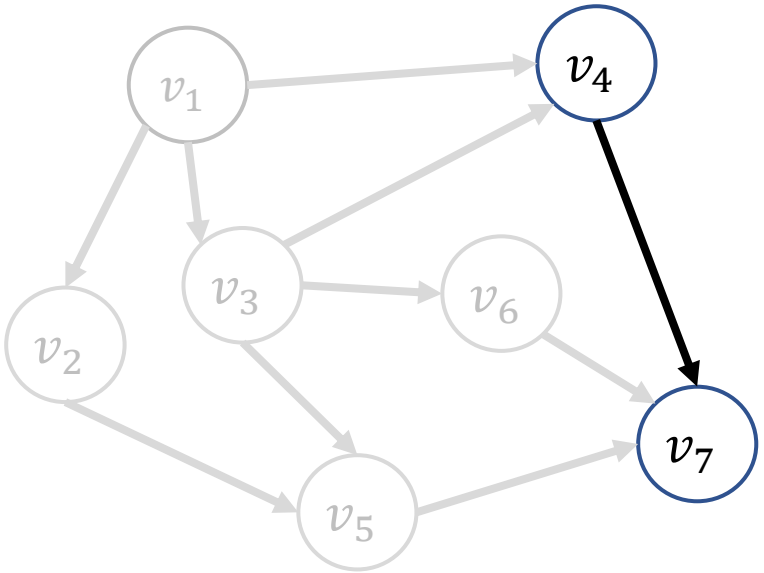


AOV 網路中的 Topological Sort (拓撲排序)

1. 從其中一個入度為 0 的頂點出發並置入排序中
2. 把該頂點的所有鄰邊刪除
3. 重複步驟 1~2，直至找不到入度為 0 的點

Topological Sort：v_1 v_2 v_3 v_6 v_5

Topological Sort

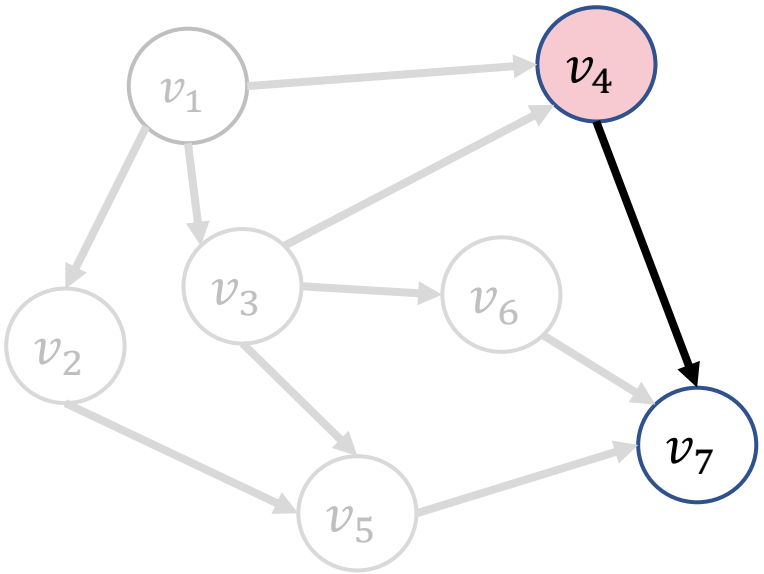


AOV 網路中的 Topological Sort (拓撲排序)

1. 從其中一個入度為 0 的頂點出發並置入排序中
2. 把該頂點的所有鄰邊刪除
3. 重複步驟 1~2，直至找不到入度為 0 的點

Topological Sort：v_1 v_2 v_3 v_6 v_5

Topological Sort

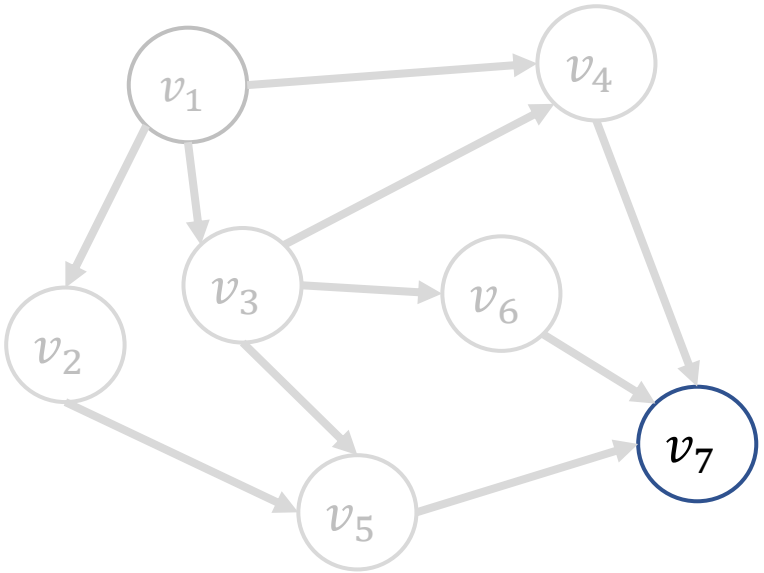


AOV 網路中的 Topological Sort (拓撲排序)

1. 從其中一個入度為 0 的頂點出發並置入排序中
2. 把該頂點的所有鄰邊刪除
3. 重複步驟 1~2，直至找不到入度為 0 的點

Topological Sort：v_1 v_2 v_3 v_6 v_5 v_4

Topological Sort

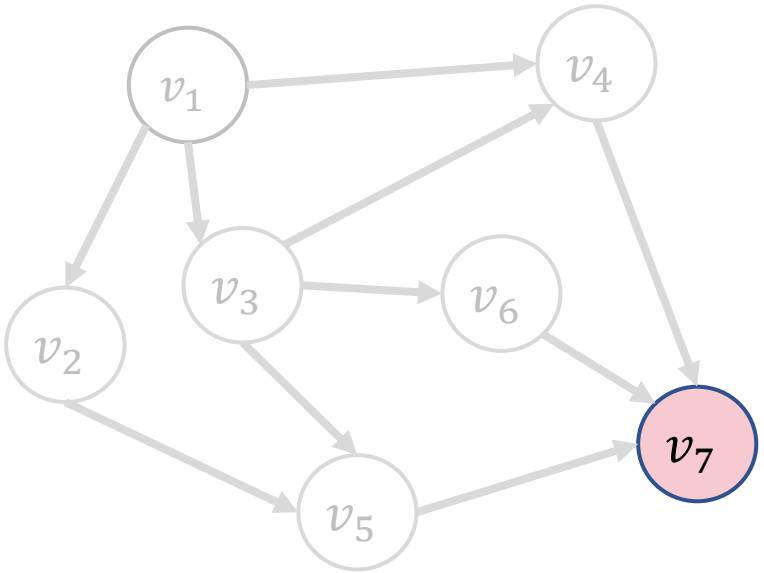


AOV 網路中的 Topological Sort (拓撲排序)

1. 從其中一個入度為 0 的頂點出發並置入排序中
2. 把該頂點的所有鄰邊刪除
3. 重複步驟 1~2，直至找不到入度為 0 的點

Topological Sort：v_1 v_2 v_3 v_6 v_5 v_4

Topological Sort

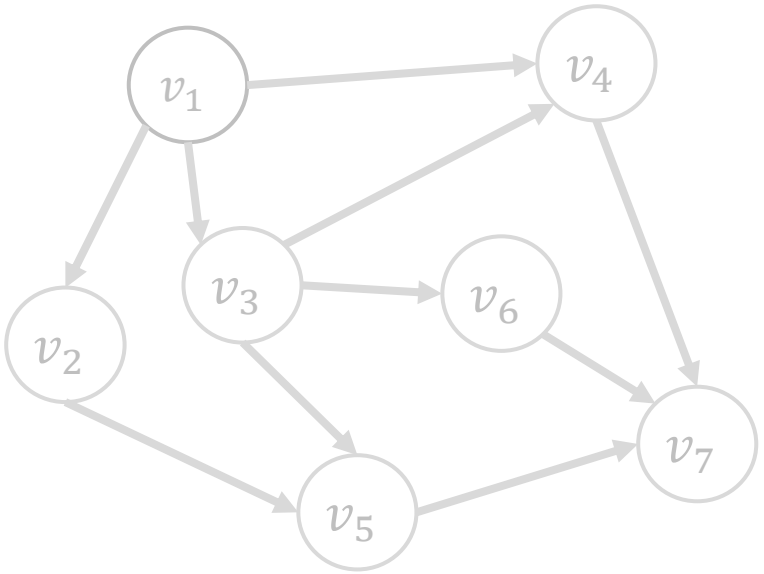


AOV 網路中的 Topological Sort (拓撲排序)

1. 從其中一個入度為 0 的頂點出發並置入排序中
2. 把該頂點的所有鄰邊刪除
3. 重複步驟 1~2，直至找不到入度為 0 的點

Topological Sort：v_1 v_2 v_3 v_6 v_5 v_4 v_7

Topological Sort

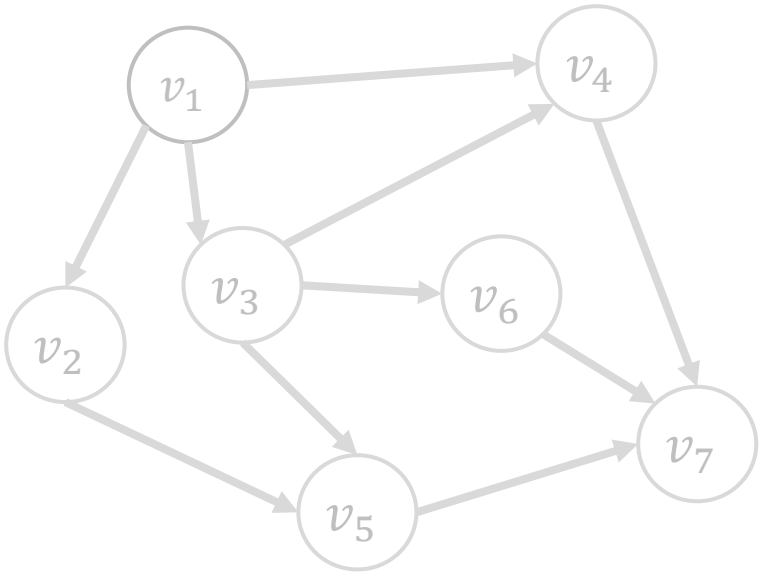


AOV 網路中的 Topological Sort (拓撲排序)

1. 從其中一個入度為 0 的頂點出發並置入排序中
2. 把該頂點的所有鄰邊刪除
3. 重複步驟 1~2，直至找不到入度為 0 的點

Topological Sort：v_1 v_2 v_3 v_6 v_5 v_4 v_7

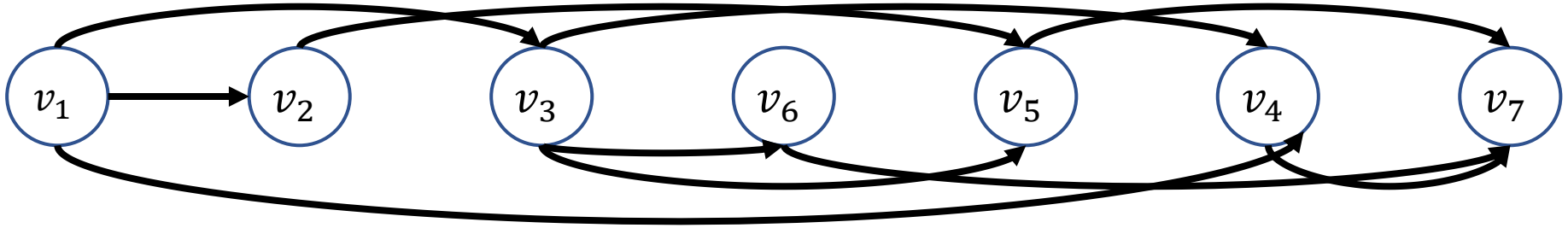
Topological Sort



AOV 網路中的 Topological Sort (拓撲排序)

1. 從其中一個入度為 0 的頂點出發並置入排序中
2. 把該頂點的所有鄰邊刪除
3. 重複步驟 1~2，直至找不到入度為 0 的點

Topological Sort：



每次有多個入度為0的點可以選擇，故Topological Sort 並非唯一解！

Practice

Mission

Try LeetCode #207. Course Schedule

There are a total of numCourses courses you have to take, labeled from 0 to numCourses - 1. You are given an array prerequisites where prerequisites[i] = [ai, bi] indicates that you must take course bi first if you want to take course ai.

- For example, the pair [0, 1], indicates that to take course 0 you have to first take course 1.

Return true if you can finish all courses. Otherwise, return false.

Ref：https://leetcode.com/problems/course-schedule/

Practice

Mission

Try LeetCode #210. Course Schedule II

There are a total of numCourses courses you have to take, labeled from 0 to numCourses - 1. You are given an array prerequisites where prerequisites[i] = [ai, bi] indicates that you must take course bi first if you want to take course ai.

For example, the pair [0, 1], indicates that to take course 0 you have to first take course 1.

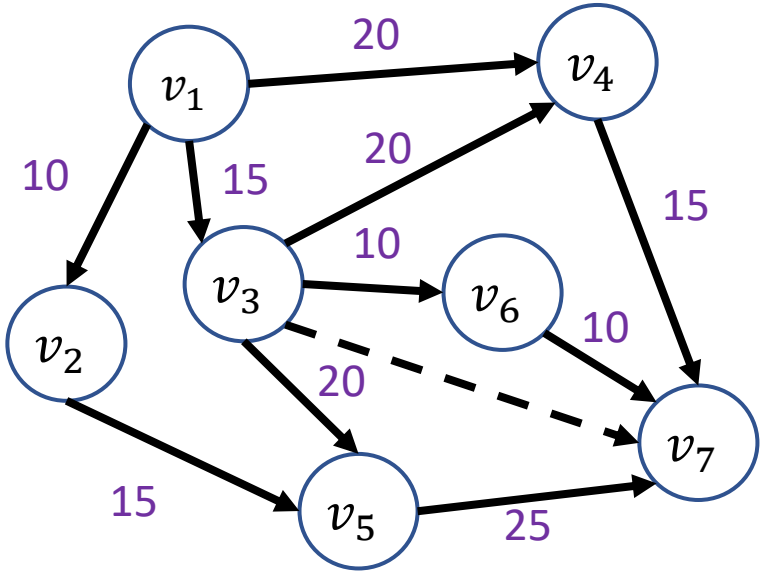
Return the ordering of courses you should take to finish all courses. If there are many valid answers, return any of them. If it is impossible to finish all courses, return an empty array.

Ref : https://leetcode.com/problems/course-schedule-ii/

AOE 網路

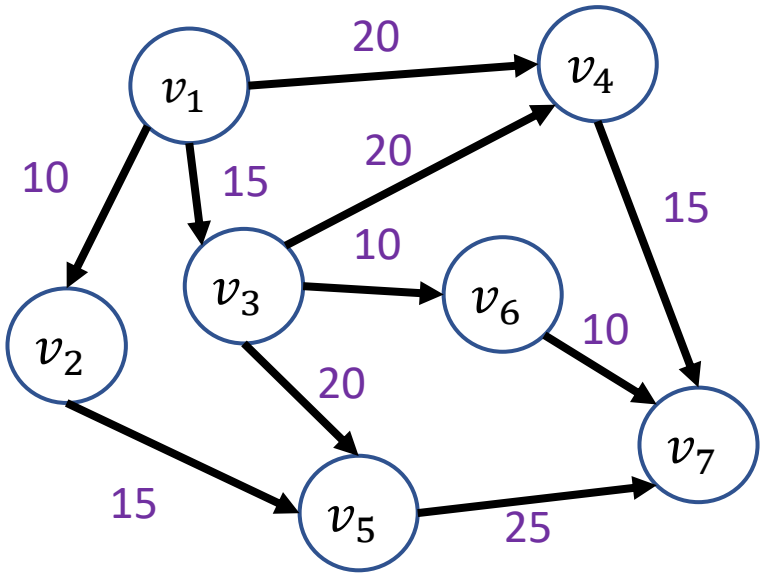
AOE 網路 (Activity On Edge)

- 強調各活動的發生順序，常用於工程或專案評估
  - ✓ 以邊表示活動 (Activity)
    - ◘ 邊上的權重代表完成該活動所需時間
  - ✓ 頂點表示事件 (Event)



- ◘ 須完成 v_1、v_3 才能進行 v_4
- ◘ $e(v_1, v_4)$ 需要 20 天、 $e(v_3, v_6)$ 需要 10 天
- ◘ $e(v_3, v_7)$ 為虛擬活動路徑 (Dummy Activity Path)
  - ✓ 在求關鍵路徑時會令 $e(v_3, v_7) = 0$

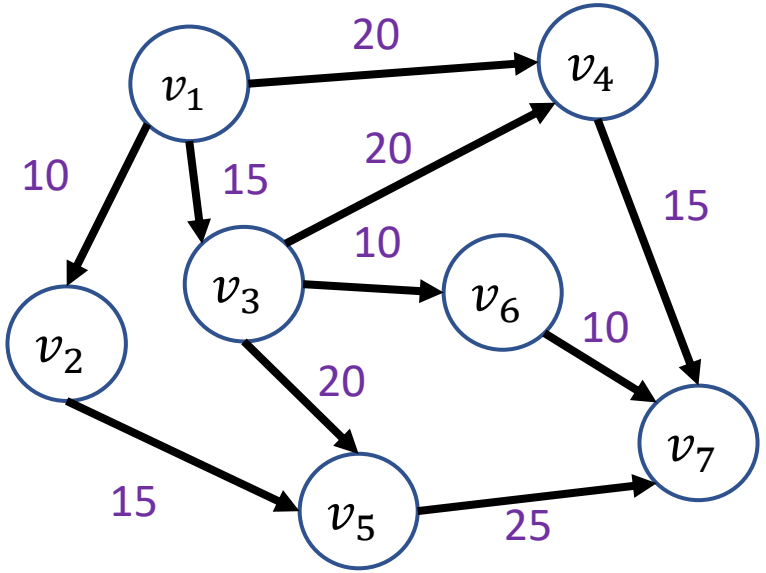
AOE 網路



AOE 網路 (Activity On Edge) 中的關鍵路徑 (Critical Path)

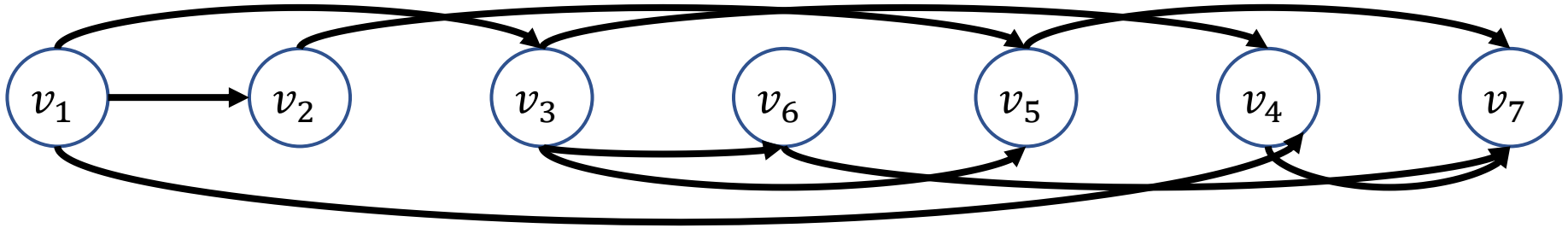
- 專案所需的最短時間，就是起點到終點的最長路徑
  - ✓ 該路徑稱為關鍵路徑 (Critical Path)
  - ✓ 關鍵路徑 (Critical Path) 可能不只一條
  - ✓ 找出關鍵路徑後就能夠集中資源並加速整個專案
- AOE 網路中所有活動皆有時間
  1. 最早開始時間 (Early time)：活動能夠最早啟動的時間
  2. 最晚開始時間 (Late time)：不影響整個專案下，該活動能夠最晚啟動的時間

AOE 網路



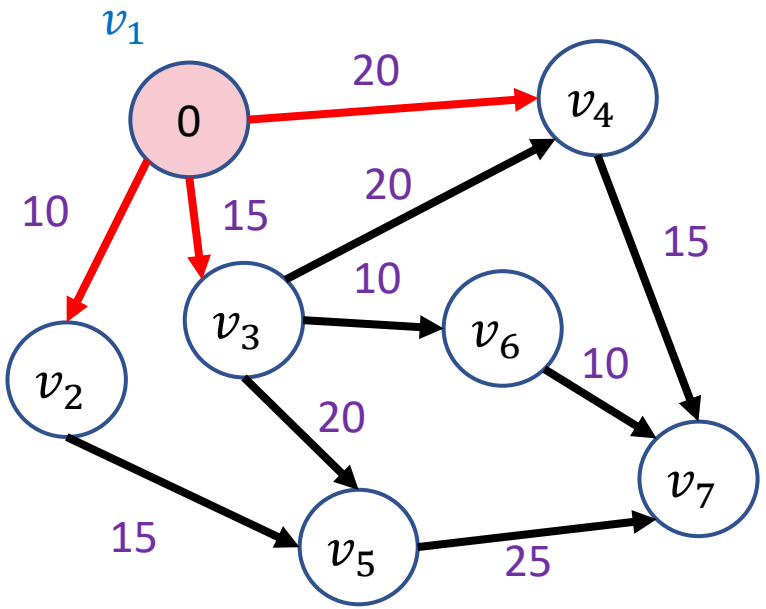
- AOE 網路中所有活動皆有時間，一開始初始化為 0
  1. 最早開始時間 (Early time)
     - ✓ $E(v_j) = max(E(v_j), E(v_i) + w(v_i, v_j))$
     - ✓ 需經拓樸排序後逐一運算

Topological Sort：



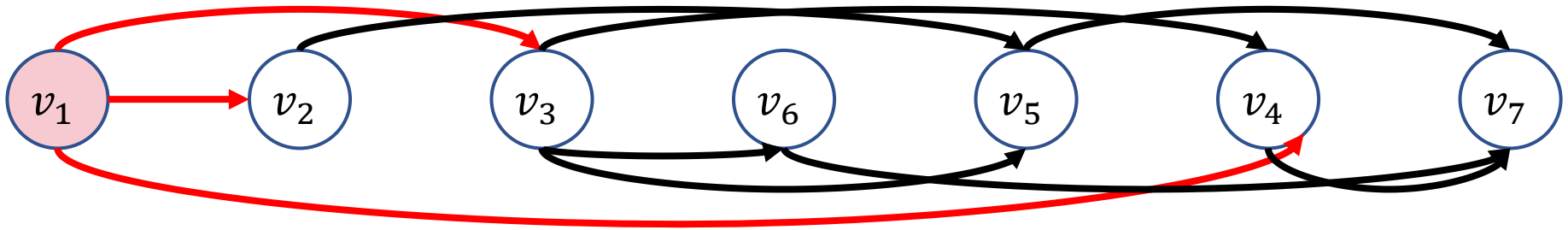
v_i	1	2	3	4	5	6	7
$E(v_i)$	0	0	0	0	0	0	0

AOE 網路



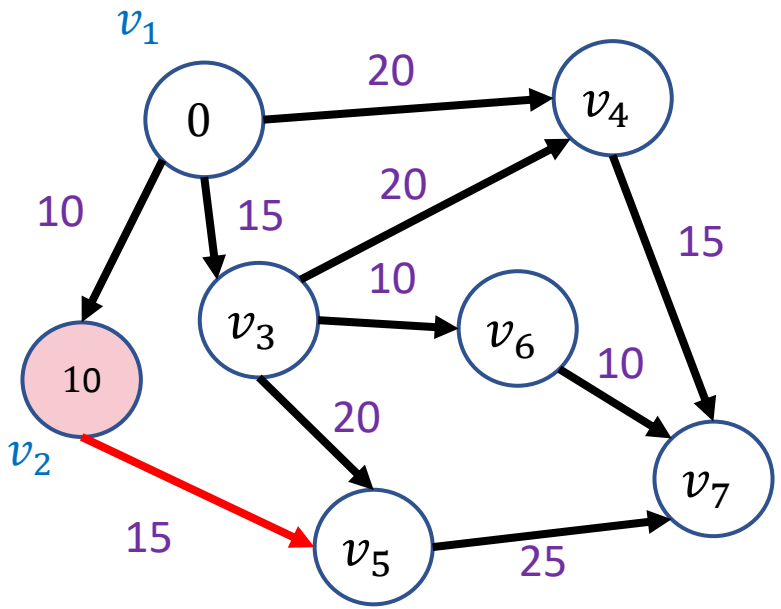
- AOE 網路中所有活動皆有時間
  1. 最早開始時間 (Early time)
     - ✓ $E(v_j) = max(E(v_j), E(v_i) + w(v_i, v_j))$
     - ✓ 需經拓樸排序後逐一運算

Topological Sort：



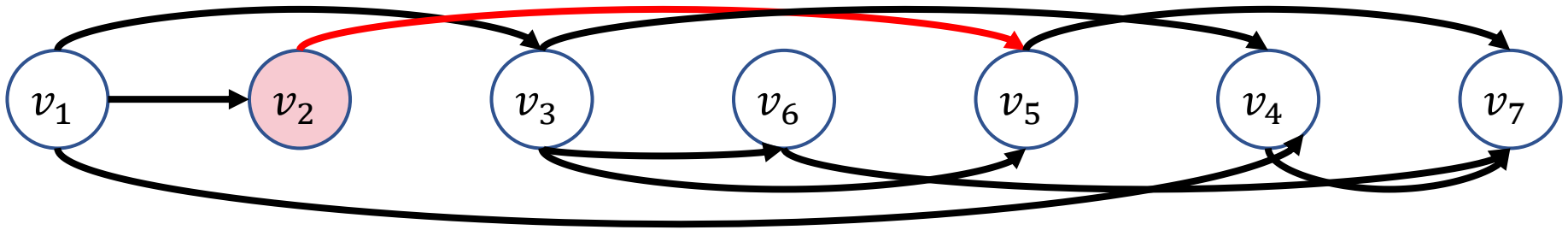
v_i	1	2	3	4	5	6	7
$E(v_i)$	0	10	15	20	0	0	0

AOE 網路



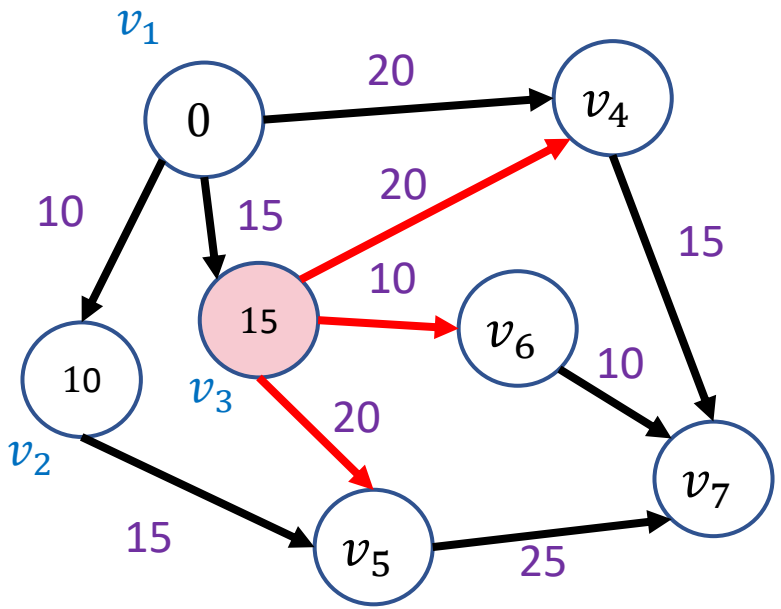
- AOE 網路中所有活動皆有時間
  1. 最早開始時間 (Early time)
     - ✓ $E(v_j) = max(E(v_j), E(v_i) + w(v_i, v_j))$
     - ✓ 需經拓樸排序後逐一運算

Topological Sort：



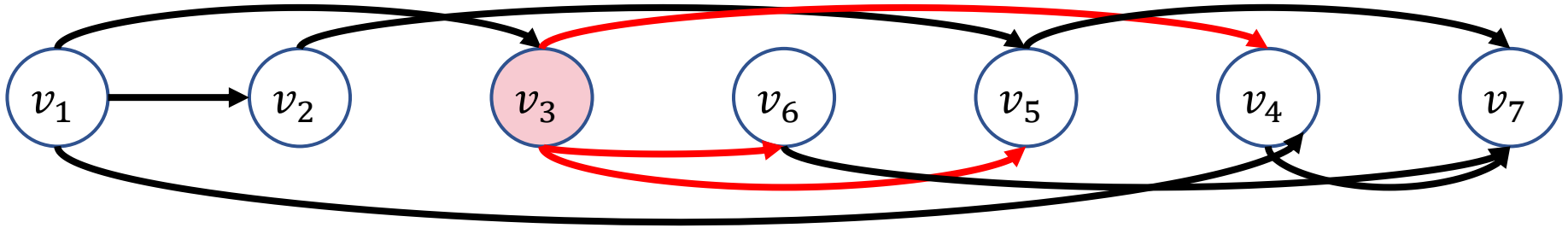
v_i	1	2	3	4	5	6	7
$E(v_i)$	0	10	15	20	25	0	0

AOE 網路



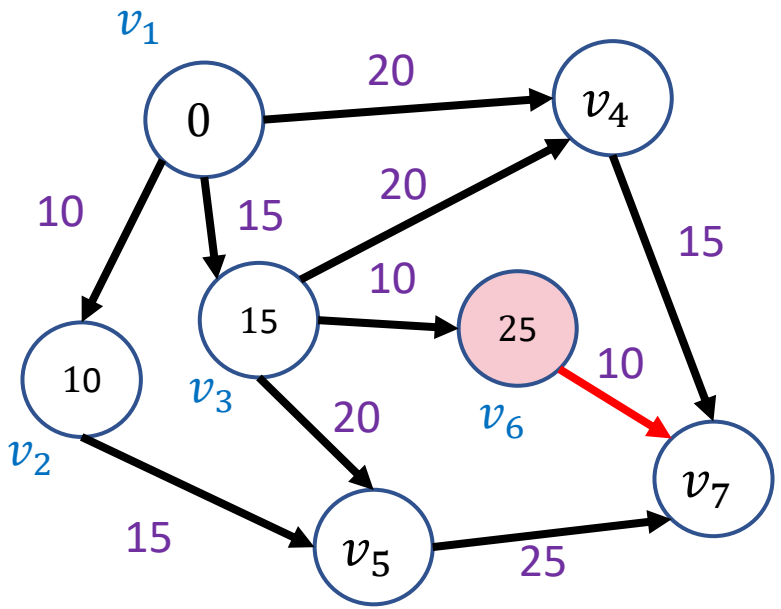
- AOE 網路中所有活動皆有時間
  1. 最早開始時間 (Early time)
     - ✓ $E(v_j) = max(E(v_j), E(v_i) + w(v_i, v_j))$
     - ✓ 需經拓樸排序後逐一運算

Topological Sort：



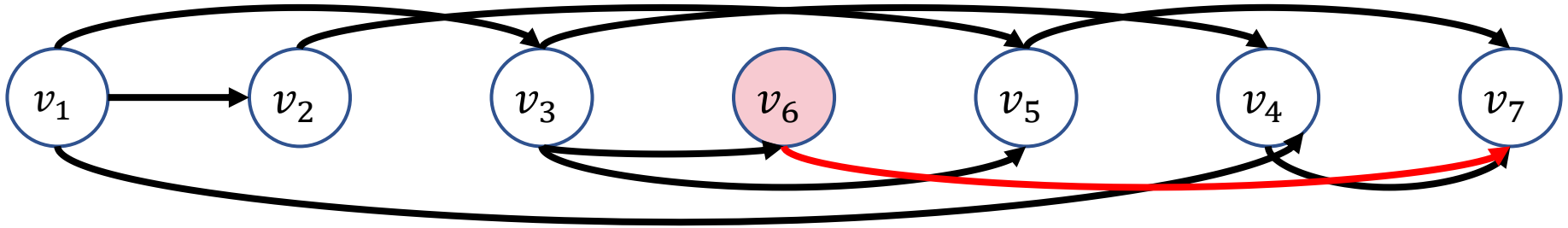
v_i	1	2	3	4	5	6	7
$E(v_i)$	0	10	15	35	35	25	0

AOE 網路



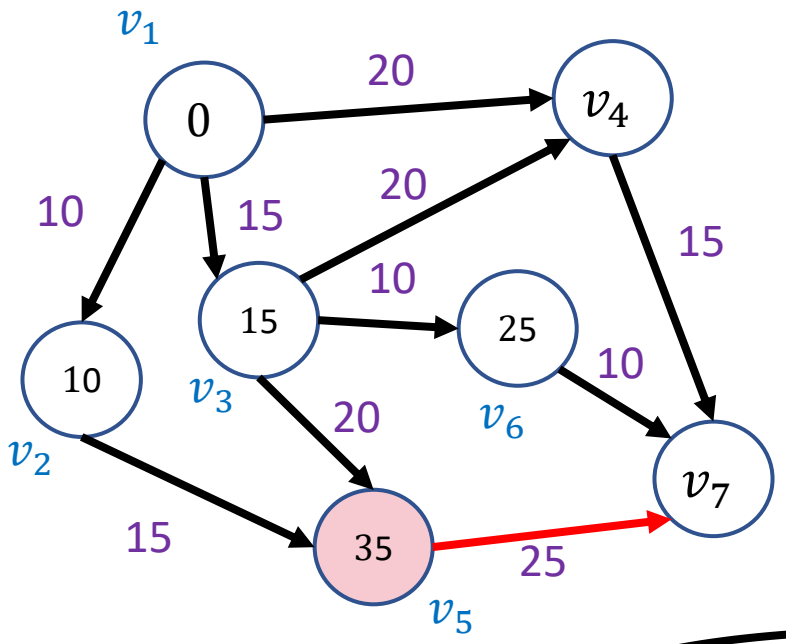
- AOE 網路中所有活動皆有時間
  1. 最早開始時間 (Early time)
     - ✓ $E(v_j) = max(E(v_j), E(v_i) + w(v_i, v_j))$
     - ✓ 需經拓樸排序後逐一運算

Topological Sort：



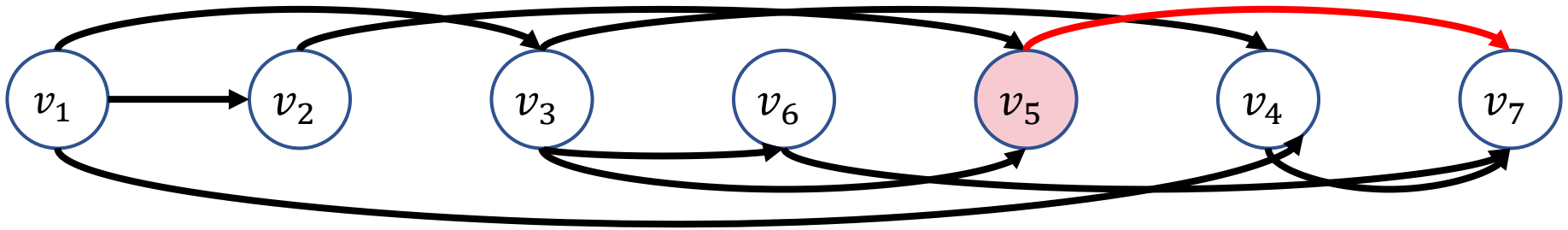
v_i	1	2	3	4	5	6	7
$E(v_i)$	0	10	15	35	35	25	35

AOE 網路



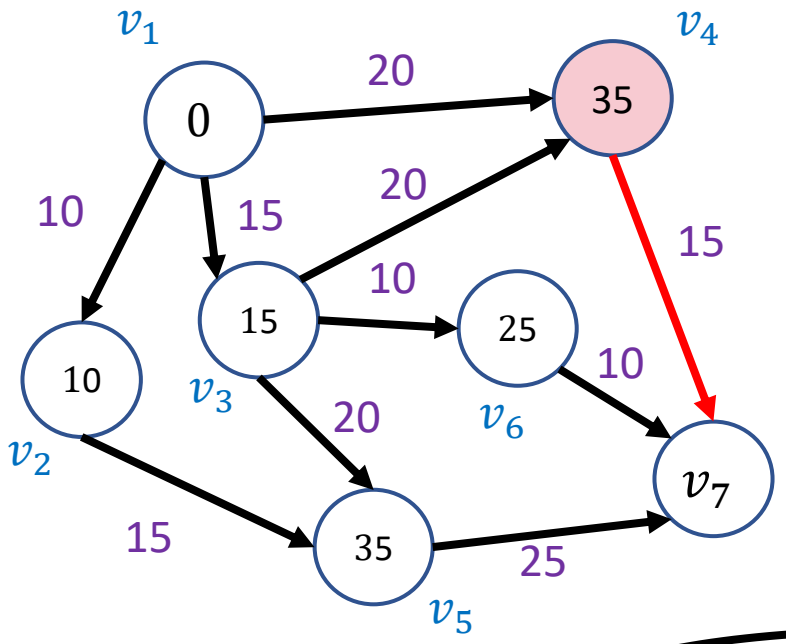
- AOE 網路中所有活動皆有時間
  1. 最早開始時間 (Early time)
     - ✓ $E(v_j) = max(E(v_j), E(v_i) + w(v_i, v_j))$
     - ✓ 需經拓樸排序後逐一運算

Topological Sort：



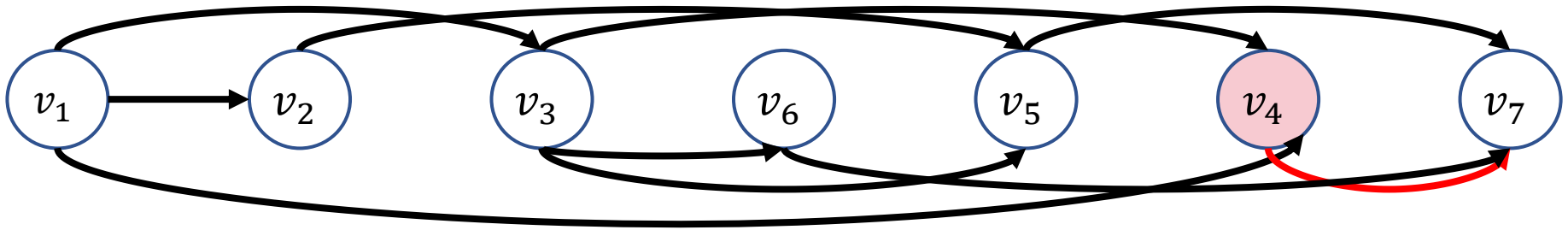
v_i	1	2	3	4	5	6	7
$E(v_i)$	0	10	15	35	35	25	60

AOE 網路



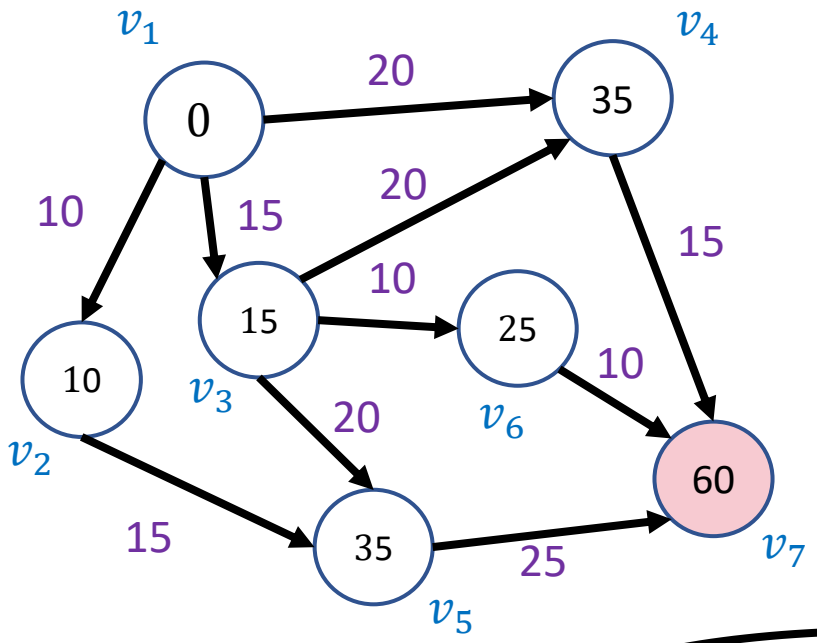
- AOE 網路中所有活動皆有時間
  1. 最早開始時間 (Early time)
     - ✓ $E(v_j) = max(E(v_j), E(v_i) + w(v_i, v_j))$
     - ✓ 需經拓樸排序後逐一運算

Topological Sort：



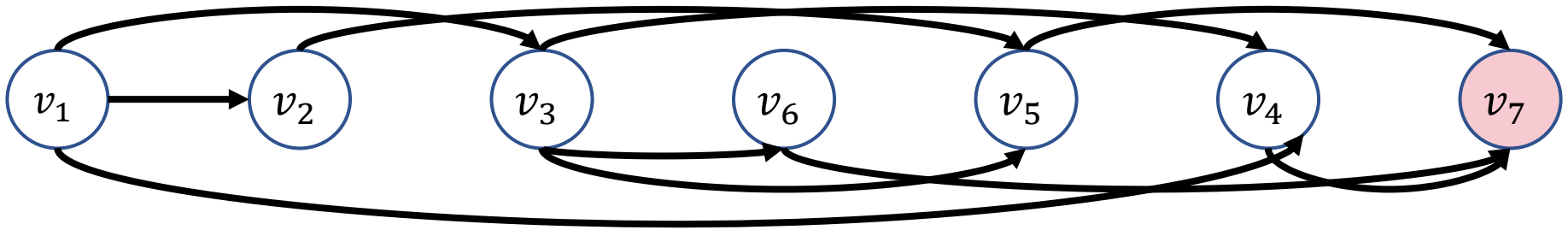
v_i	1	2	3	4	5	6	7
$E(v_i)$	0	10	15	35	35	25	60

AOE 網路



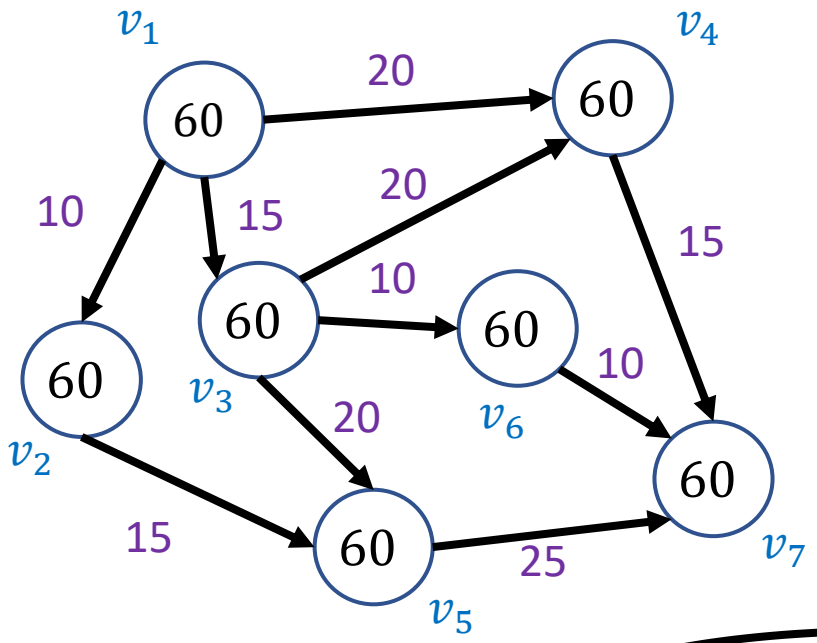
- AOE 網路中所有活動皆有時間
  1. 最早開始時間 (Early time)
     - ✓ $E(v_j) = max(E(v_j), E(v_i) + w(v_i, v_j))$
     - ✓ 需經拓樸排序後逐一運算

Topological Sort：



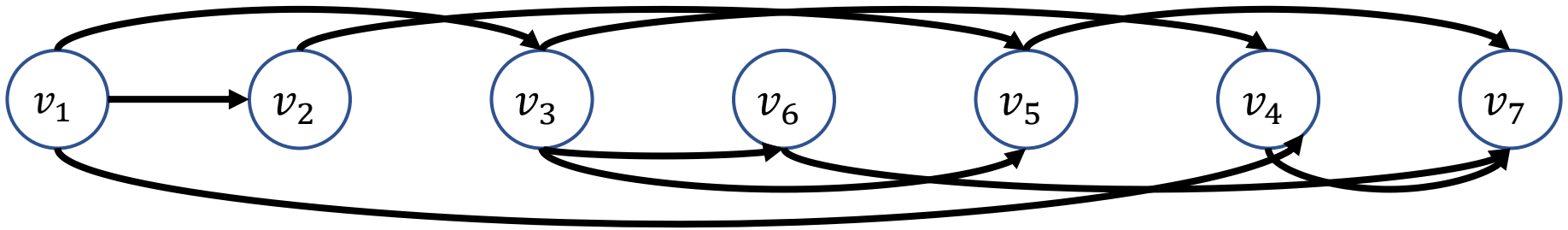
v_i	1	2	3	4	5	6	7
$E(v_i)$	0	10	15	35	35	25	60

AOE 網路



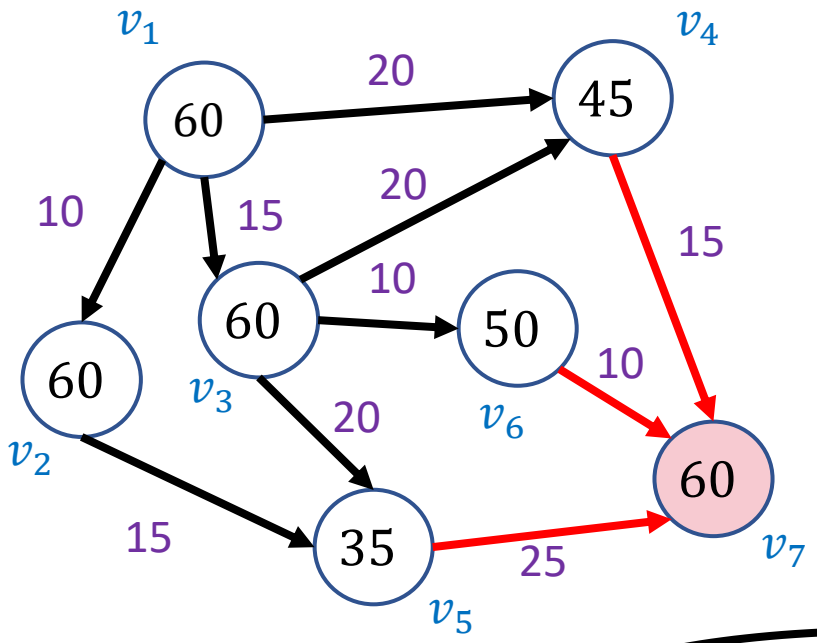
- AOE 網路中所有活動皆有時間，一開始初始化為：
  - ✓ 終點的最早開始時間 (Early time)
  - 2. 最晚開始時間 (Late time)
    - ✓ $L(v_j) = min(L(v_j), L(v_i) - w(v_i, v_j))$
    - ✓ 需經反向拓樸排序後逐一運算

Topological Sort：



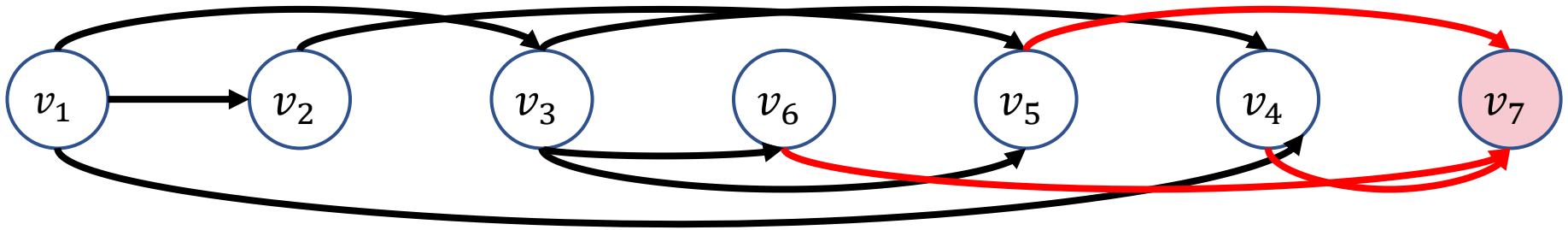
v_i	1	2	3	4	5	6	7
$E(v_i)$	60	60	60	60	60	60	60

AOE 網路



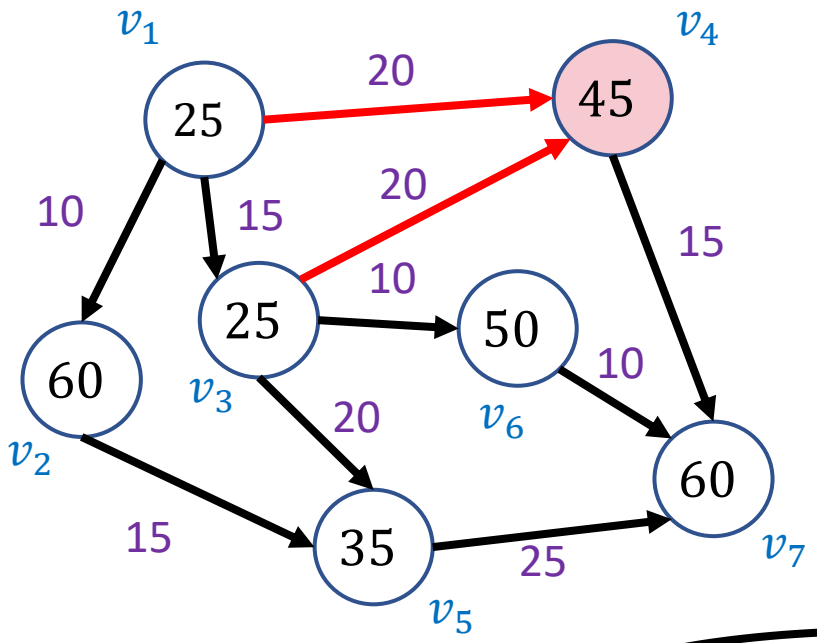
- AOE 網路中所有活動皆有時間，一開始初始化為：
  - ✓ 終點的最早開始時間 (Early time)
  - 2. 最晚開始時間 (Late time)
    - ✓ $L(v_j) = min(L(v_j), L(v_i) - w(v_i, v_j))$
    - ✓ 需經反向拓樸排序後逐一運算

Topological Sort：



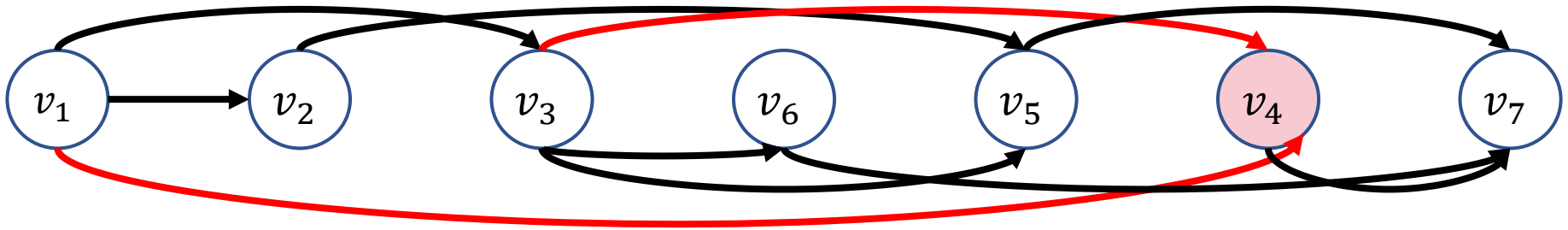
v_i	1	2	3	4	5	6	7
$E(v_i)$	60	60	60	45	35	50	60

AOE 網路



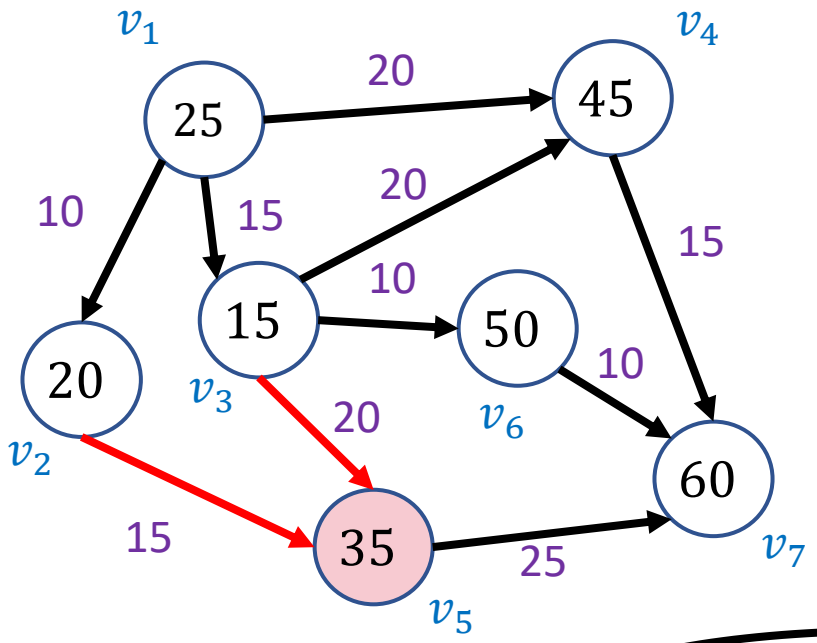
- AOE 網路中所有活動皆有時間，一開始初始化為：
  - ✓ 終點的最早開始時間 (Early time)
  - 2. 最晚開始時間 (Late time)
    - ✓ $L(v_j) = min(L(v_j), L(v_i) - w(v_i, v_j))$
    - ✓ 需經反向拓樸排序後逐一運算

Topological Sort：



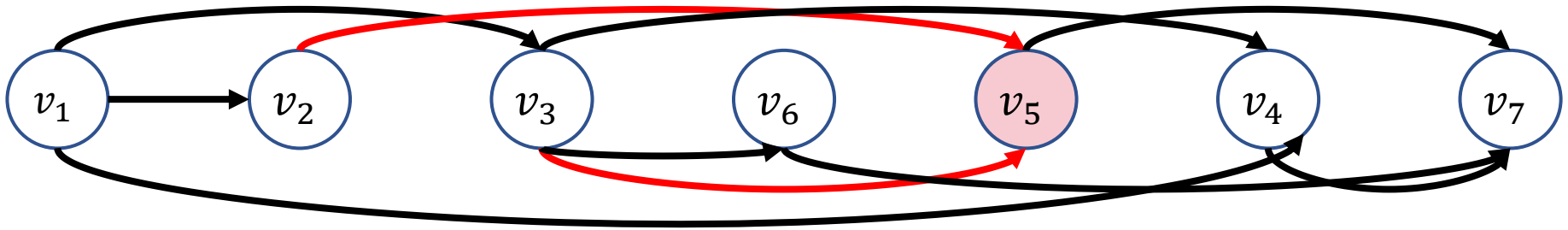
v_i	1	2	3	4	5	6	7
$E(v_i)$	25	60	25	45	35	50	60

AOE 網路



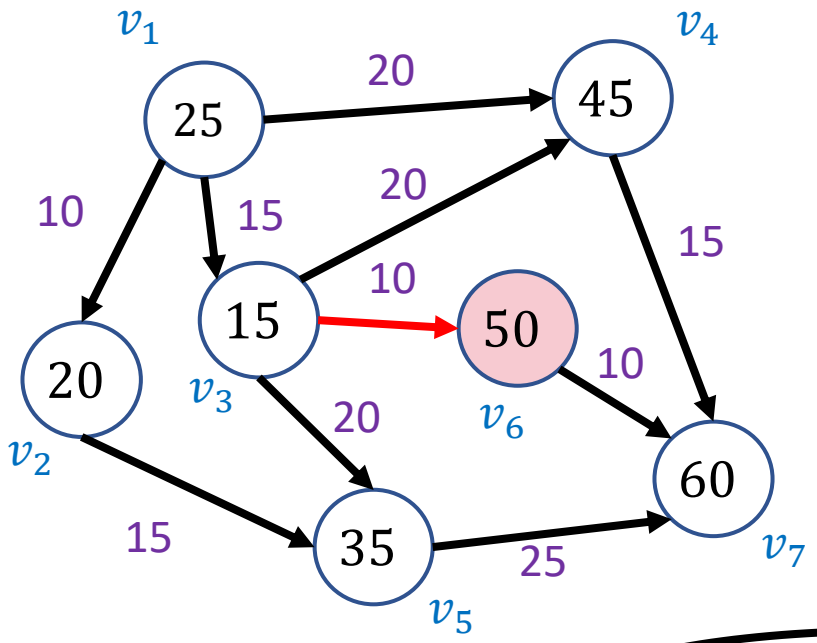
- AOE 網路中所有活動皆有時間，一開始初始化為：
  - ✓ 終點的最早開始時間 (Early time)
  - 2. 最晚開始時間 (Late time)
    - ✓ $L(v_j) = min(L(v_j), L(v_i) - w(v_i, v_j))$
    - ✓ 需經反向拓樸排序後逐一運算

Topological Sort：



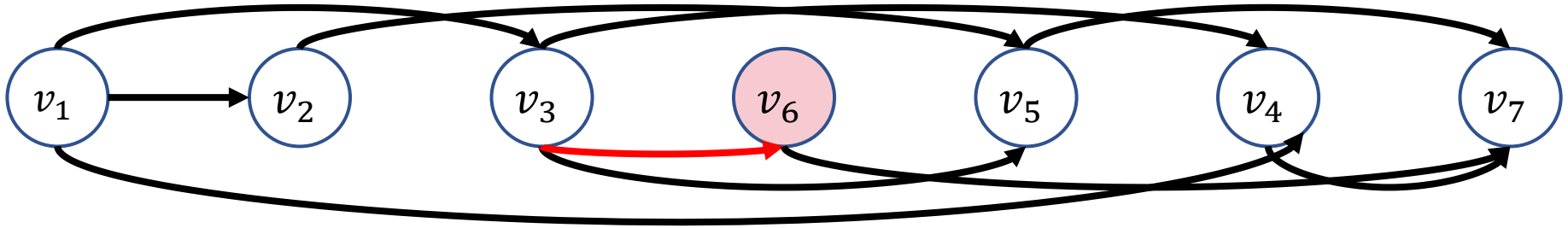
v_i	1	2	3	4	5	6	7
$E(v_i)$	25	20	15	45	35	50	60

AOE 網路



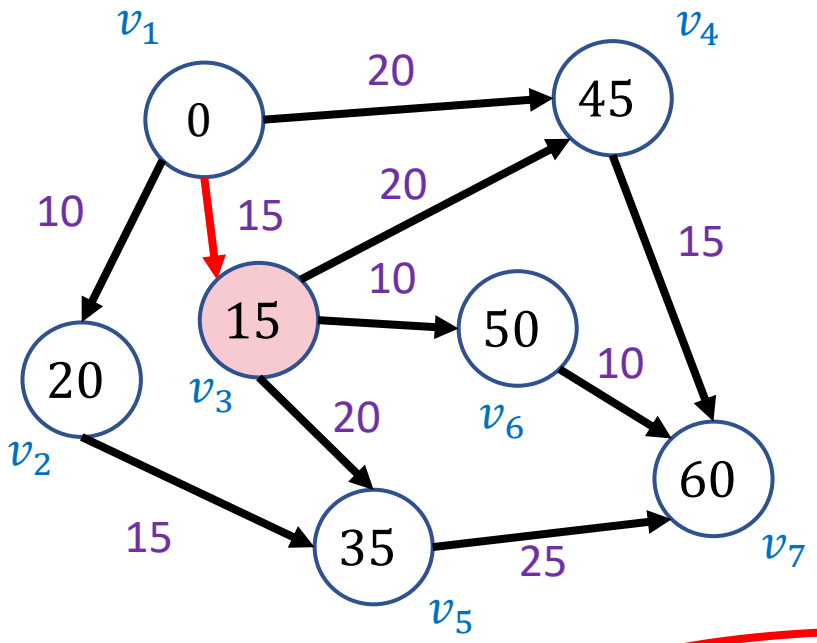
- AOE 網路中所有活動皆有時間，一開始初始化為：
  - ✓ 終點的最早開始時間 (Early time)
  - 2. 最晚開始時間 (Late time)
    - ✓ $L(v_j) = min(L(v_j), L(v_i) - w(v_i, v_j))$
    - ✓ 需經反向拓樸排序後逐一運算

Topological Sort：



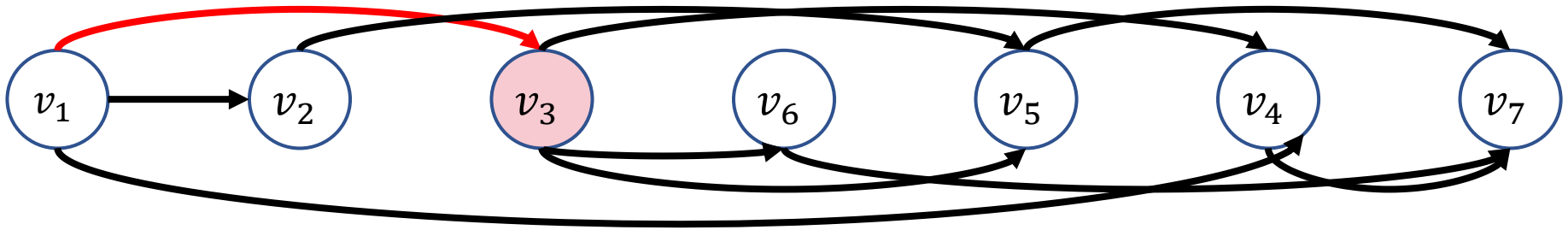
v_i	1	2	3	4	5	6	7
$E(v_i)$	25	20	15	45	35	50	60

AOE 網路



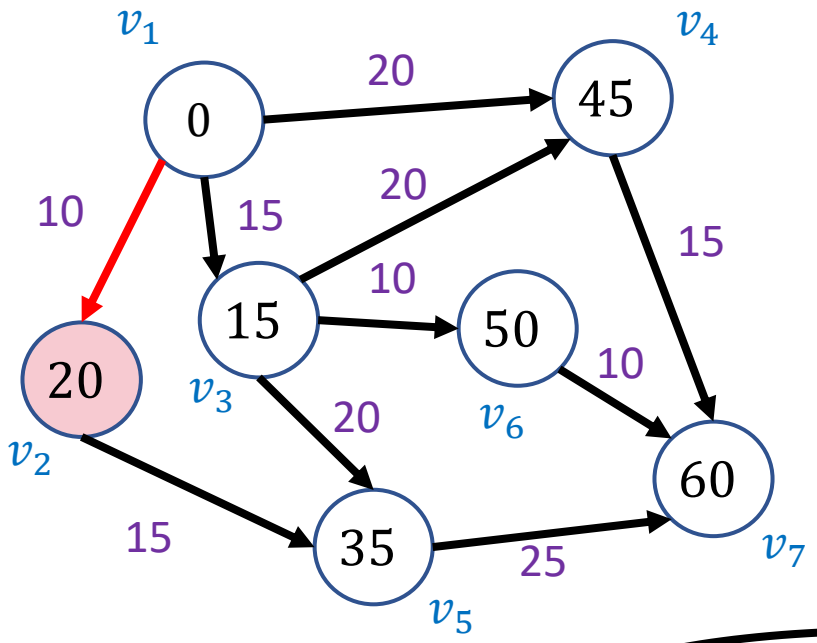
- AOE 網路中所有活動皆有時間，一開始初始化為：
  - ✓ 終點的最早開始時間 (Early time)
  - 2. 最晚開始時間 (Late time)
    - ✓ $L(v_j) = min(L(v_j), L(v_i) - w(v_i, v_j))$
    - ✓ 需經反向拓樸排序後逐一運算

Topological Sort：



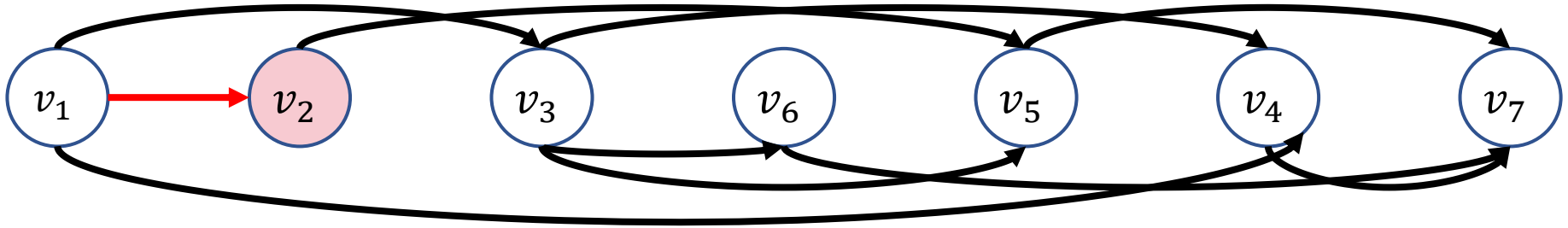
v_i	1	2	3	4	5	6	7
$E(v_i)$	0	20	15	45	35	50	60

AOE 網路



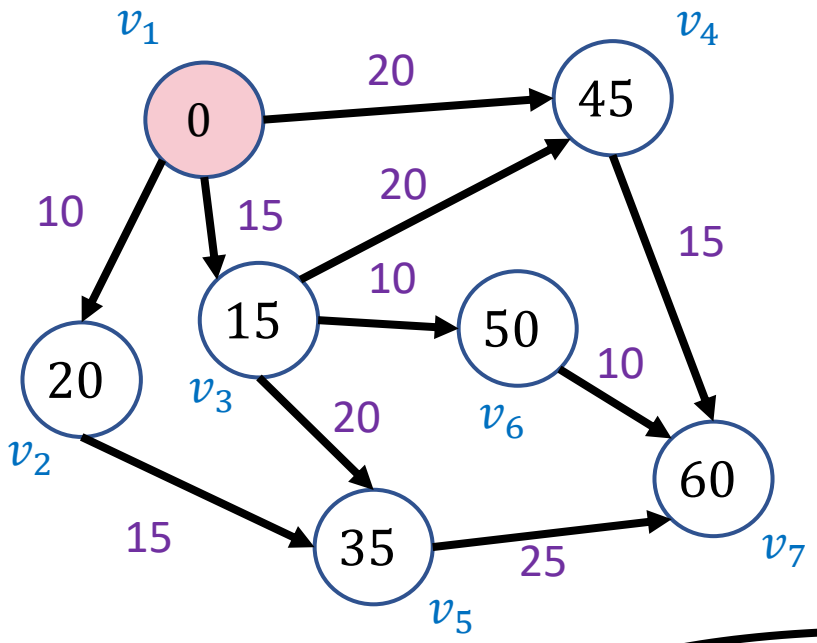
- AOE 網路中所有活動皆有時間，一開始初始化為：
  - ✓ 終點的最早開始時間 (Early time)
  - 2. 最晚開始時間 (Late time)
    - ✓ $L(v_j) = min(L(v_j), L(v_i) - w(v_i, v_j))$
    - ✓ 需經反向拓樸排序後逐一運算

Topological Sort：



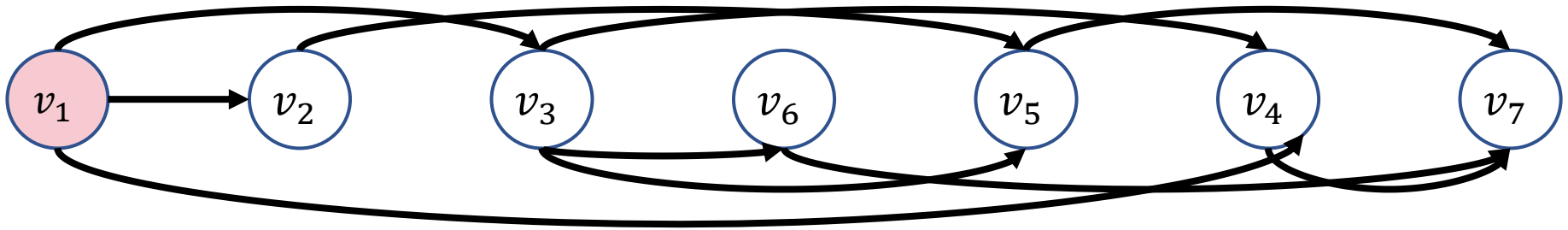
v_i	1	2	3	4	5	6	7
$E(v_i)$	0	20	15	45	35	50	60

AOE 網路



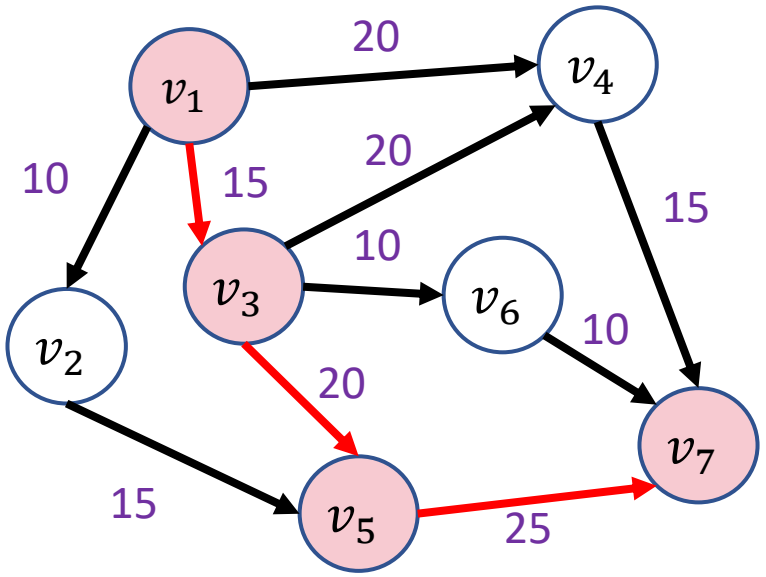
- AOE 網路中所有活動皆有時間，一開始初始化為：
  - ✓ 終點的最早開始時間 (Early time)
  - 2. 最晚開始時間 (Late time)
    - ✓ $L(v_j) = min(L(v_j), L(v_i) - w(v_i, v_j))$
    - ✓ 需經反向拓樸排序後逐一運算

Topological Sort：



v_i	1	2	3	4	5	6	7
$E(v_i)$	0	20	15	45	35	50	60

AOE 網路



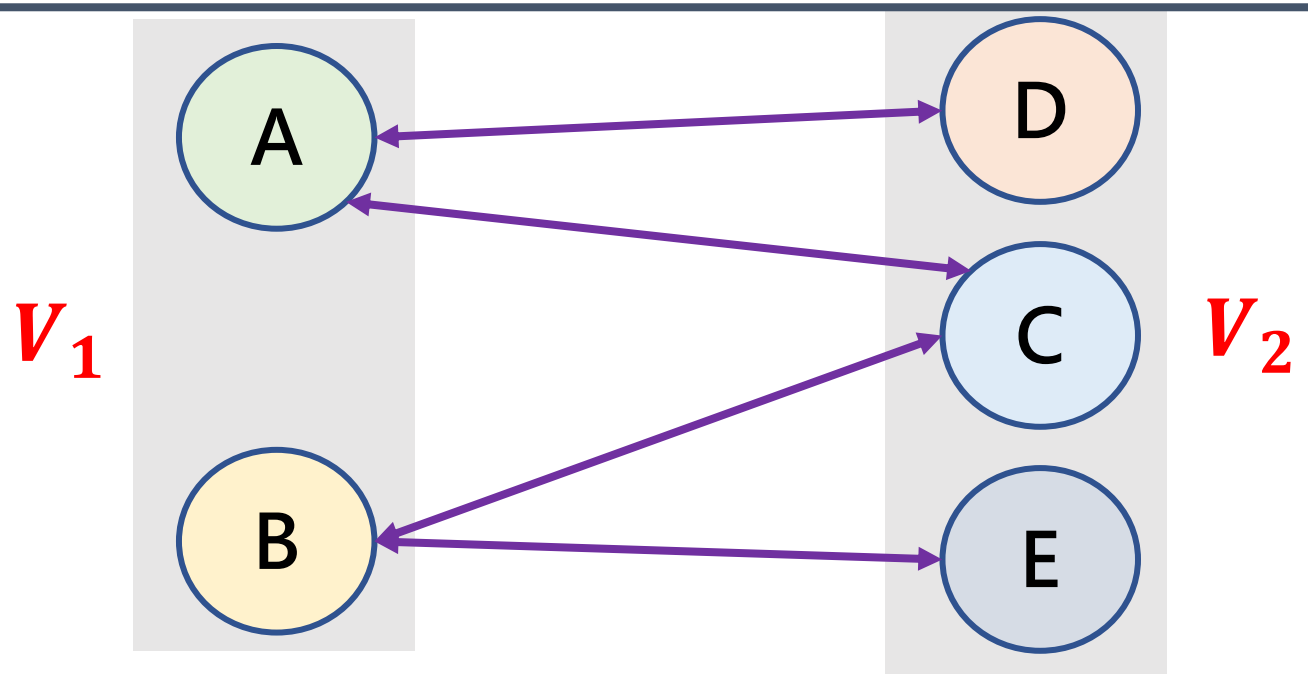
AOE 網路 (Activity On Edge) 中的關鍵路徑 (Critical Path)

- 專案所需的最短時間，就是起點到終點的最長路徑
  - ✓ 該路徑稱為關鍵路徑 (Critical Path)
  - ✓ $L(v_i) - E(v_i)$ 表 v_i 最多延後多少時間不會影響整個專案
    - ☐ $L(v_i) - E(v_i) = 0$時 v_i 為關鍵活動 (Critical Activity)
- 滿足以下三條件時，邊 $e(v_i, v_j)$ 為關鍵路徑
  1. $E(v_i) = L(v_i)$
  2. $E(v_j) = L(v_j)$
  3. $E(v_j) - E(v_i) = L(v_j) - L(v_i) = w(v_i, v_j)$

v_i	1	2	3	4	5	6	7
$E(v_i)$	0	10	15	35	35	25	60
$L(v_i)$	0	20	15	45	35	50	60

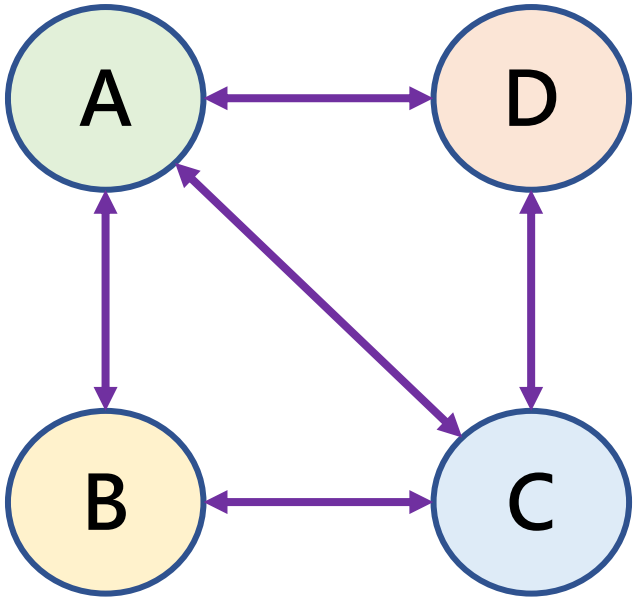
二分圖 (Bipartite Graph)

- 二分圖 (Bipartite Graph)
  - 無向圖滿足可以把頂點分成兩集合 V_1, V_2
  - 使得所有邊 $e(v_1, v_2)$ 中 v_1, v_2 必不在同一集合
  - 或無向圖進行黑白染色後，所有黑白點不相鄰



平面圖 (Plannar Graph)

- 平面圖 (Plannar Graph)
  - 可以畫在平面上，且所有的邊都不相交
  - 圖的邊可以把平面切出 F 個區塊
  - Euler 定理： $|V| - |E| + F = 2$

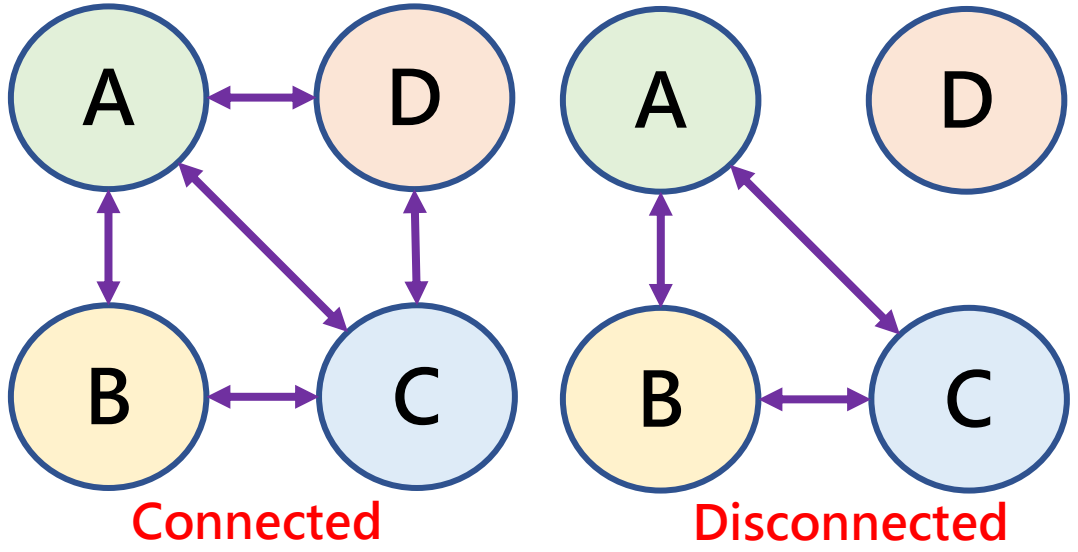


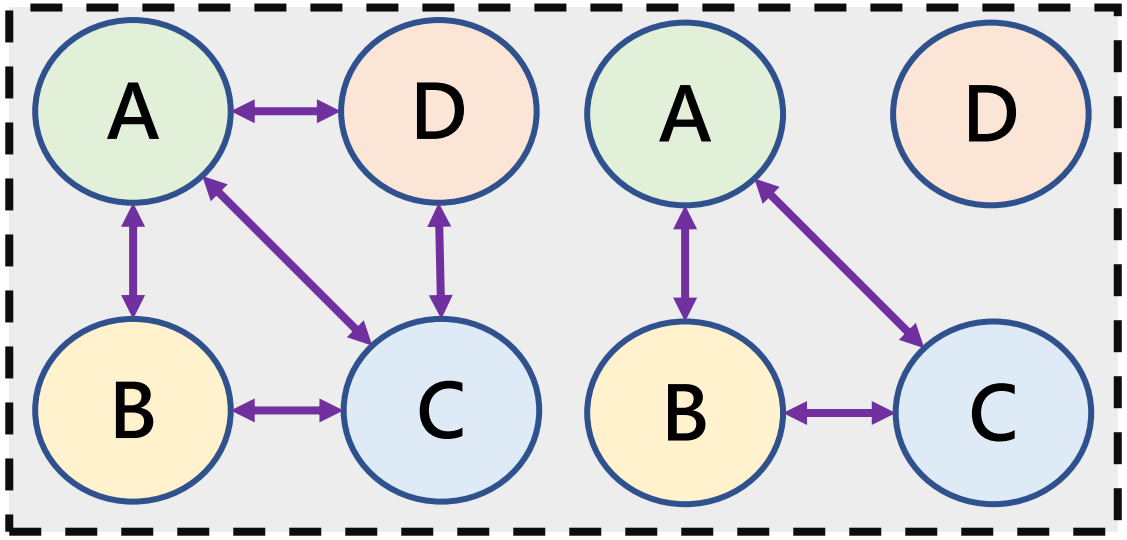
$$|V| - |E| + F = 2$$

$$4 - 5 + 3 = 2$$

連通圖 (Connected Graph)

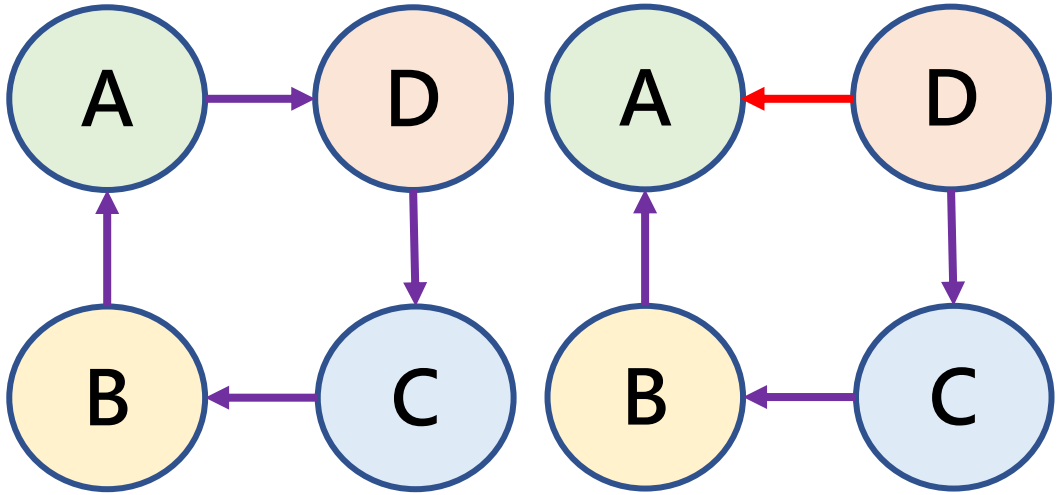
- 連通圖 (Connected Graph)
  - ➢ 無向圖 (Undirected Graph)
    - ✓ 任兩頂點間必存在一路徑
    - ✓ 否則則為 disconnected
- Connected Component
  - ➢ 最大連接後的子圖
  - ➢ 右下角的 Graph
    - ✓ 三個 Connected Component





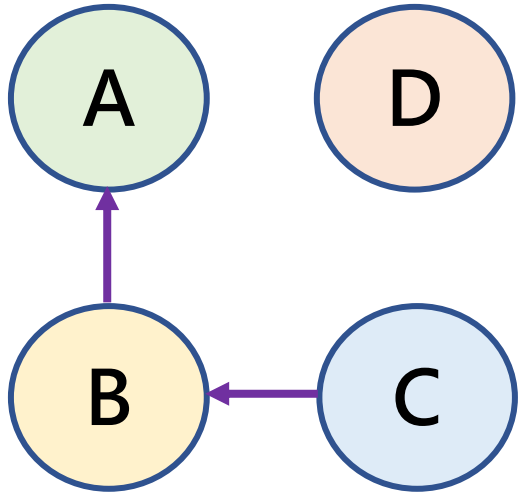
連通圖 (Connected Graph)

- 連通圖 (Connected Graph)
  - ➢ 有向圖 (Directed Graph)
    1. Strongly Connected
       - ✓ 任兩點必存在一路徑
    2. Weekly Connected
       - ✓ 化為無向圖後，任兩點必存在一路徑
    3. Disconnected
       - ✓ 非 Strongly Connected
       - ✓ 非 Weekly Connected



Strongly Connected

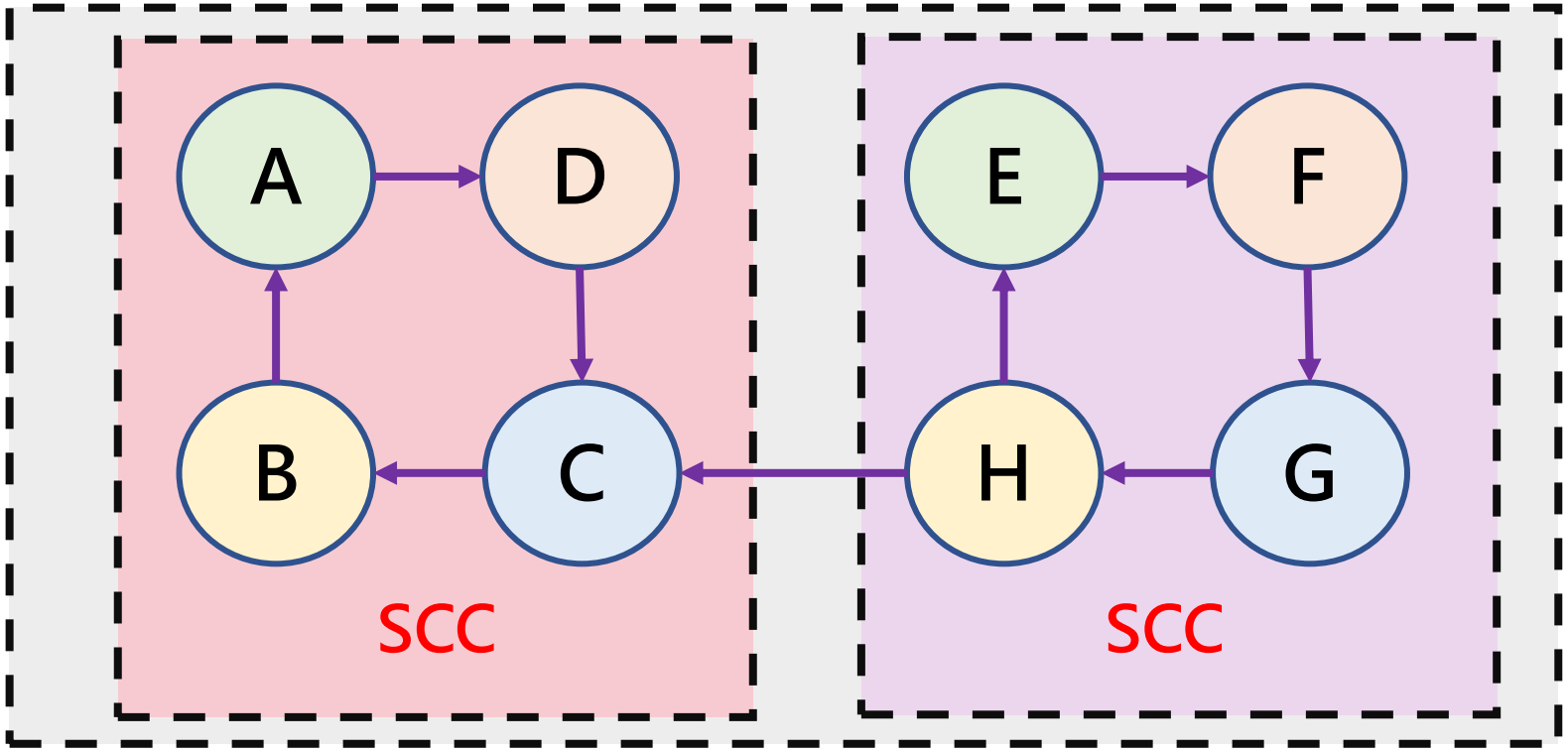
Weekly Connected



Disconnected

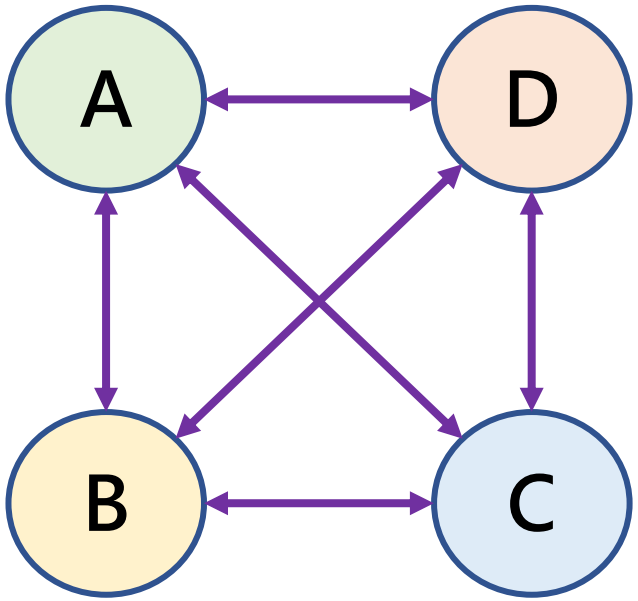
連通圖 (Connected Graph)

- 強連通元件 (Strong Connected Component，SCC)
  - 任兩點 (u,v) 間必存在 u→v 及 v→u 的路徑
  - SCC 可以組成極大強連通子圖



完全圖 (Complete Graph)

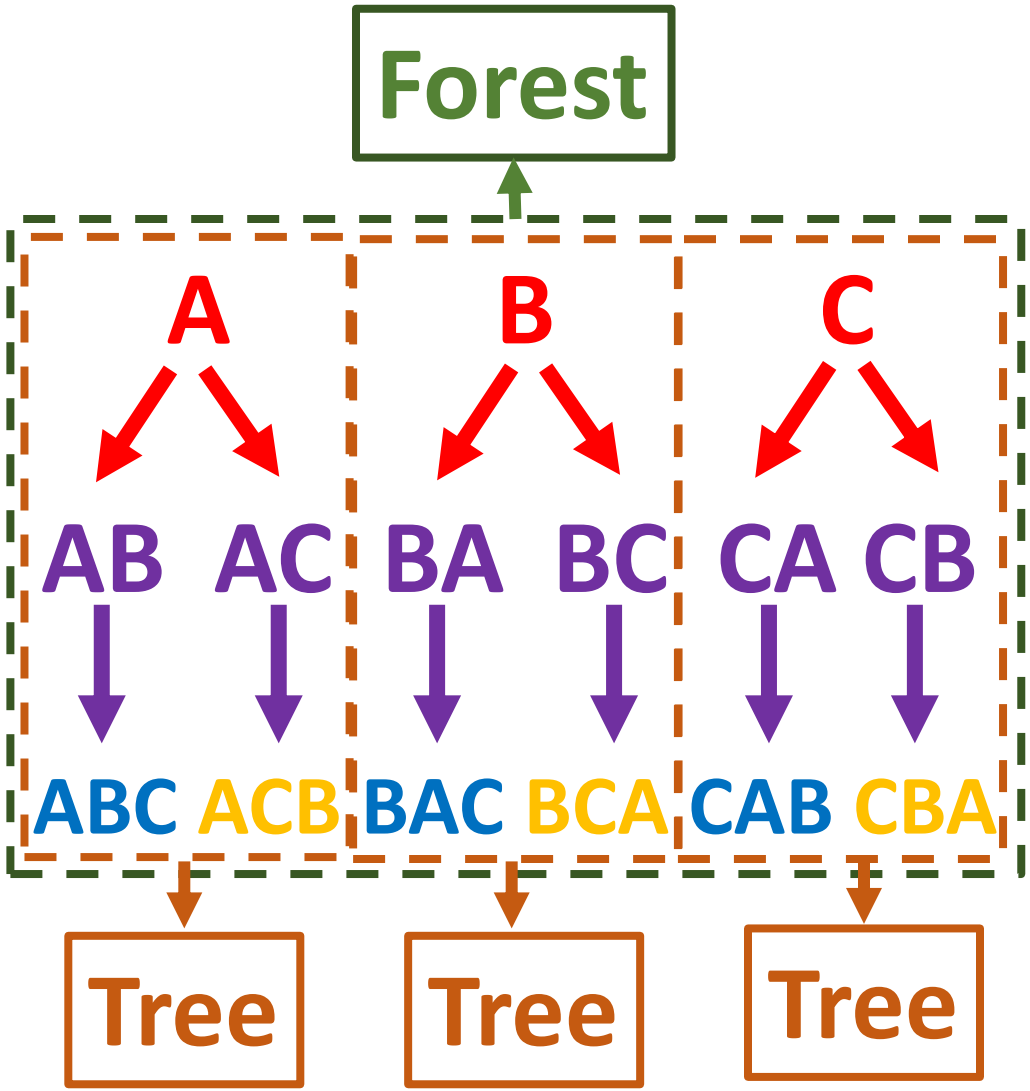
- 完全圖 (Complete Graph)
  - 所有兩兩頂點間都相鄰
  - 無向圖
    - ✓ 邊的數目：$|E| = \frac{|V| \times (|V|-1)}{2}$
    - ✓ 若 E < $\frac{|V| \times (|V|-1)}{2}$ ：not complete
  - 有向圖
    - ✓ 邊的數目：$|E| = |V| \times (|V| - 1)$
    - ✓ 若 E < $|V| \times (|V| - 1)$ ：not complete



Complete Graph

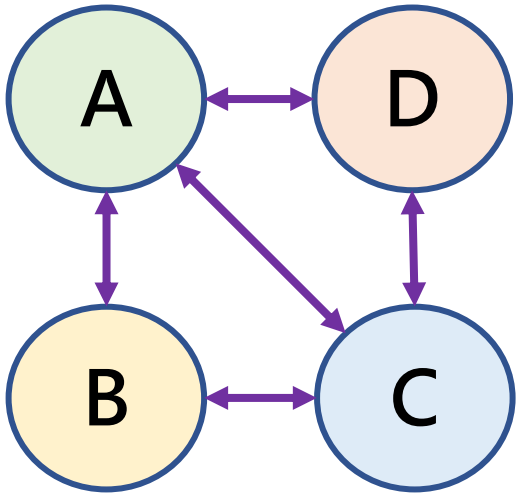
Tree & Forest

- Tree
  1. 沒有環 (Cycle)
  2. $|E| = |V| - 1$
  3. 兩兩頂點間只有一條路徑
- Forest
  - 所有的 Connected Components 都是樹

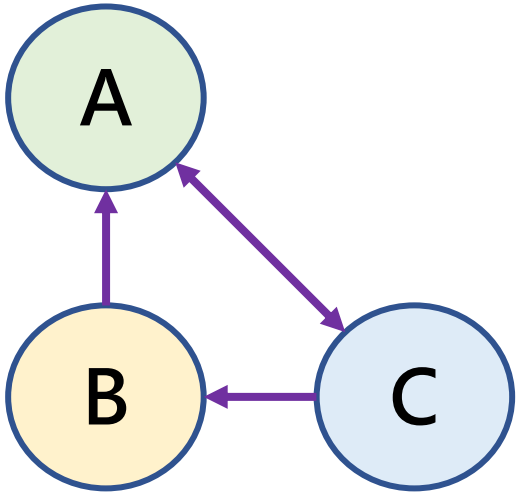


子圖、補圖、生成樹

- 子圖 (Subgraph)
  - 由圖 G 中取出子集合 G'
  - G' 即為 G 的子圖
    - ✓ G' 的 Vertex(V')亦為 G 的 Vertes(V) 的子集合
    - ✓ G' 的 Edge(E')亦為 G 的 Edge(E) 的子集合
  - 完全子圖 (Clique)
    - ✓ G' 為完全圖形



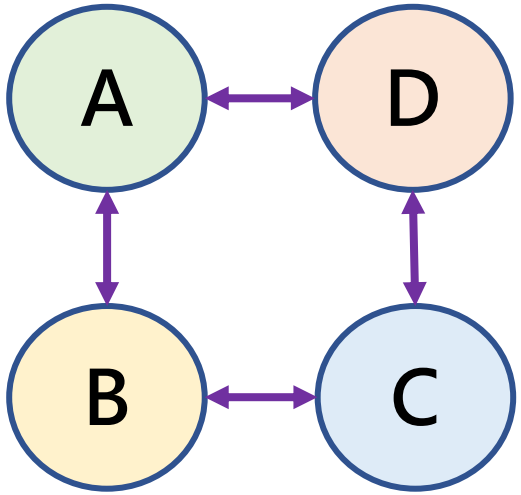
Graph



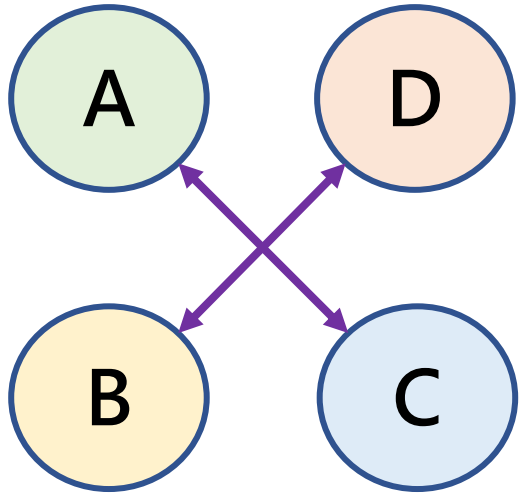
Subgraph
Clique

子圖、補圖、生成樹

- 補圖 (Compilment Graph)
  - 由圖 G' 擁有跟圖 G 相同的頂點、不同的邊
  - $V(G') = V(G); e \in E(G') \leftrightarrow e \notin E(G)$
  - 圖 G' 為圖 G 的補圖



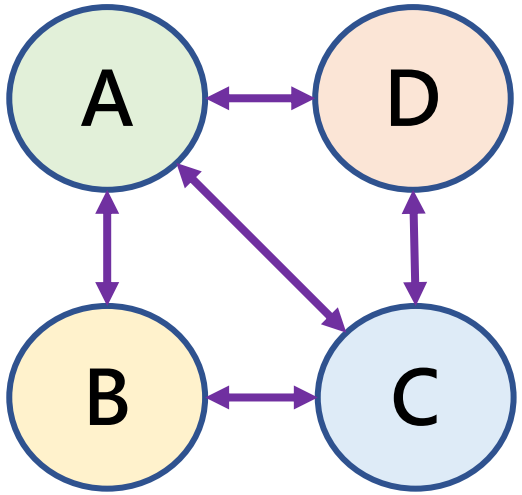
G



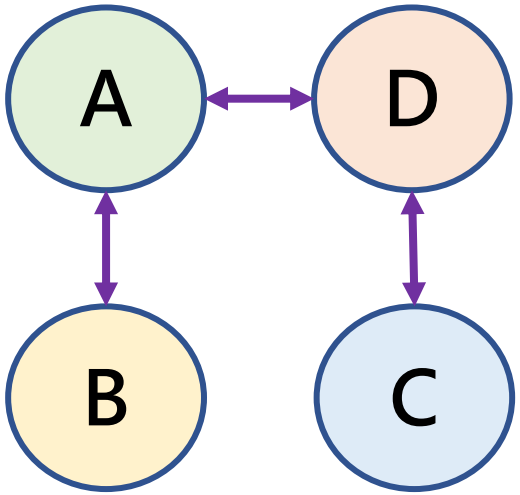
G′

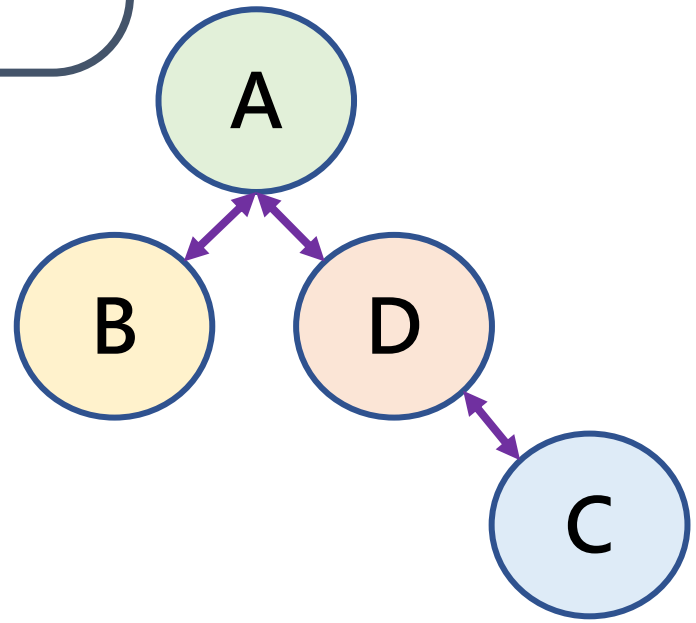
子圖、補圖、生成樹

- 生成樹 (Spanning tree)
  1. $V(G) = V(T)$
  2. 且 T 是一棵樹
  3. T 為圖 G 的生成樹



Graph

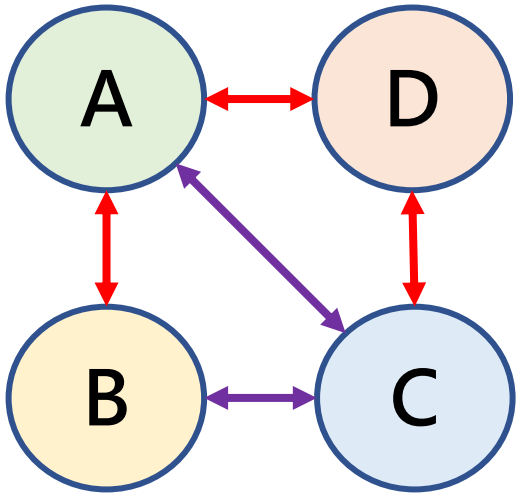




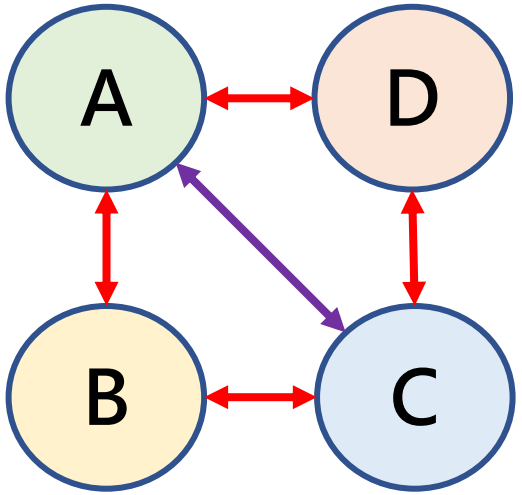
Spanning tree

Hamiltonian Path and Cycle

- 漢米爾頓路徑 (Hamiltonian Path)
  - 圖上一條簡單路徑，經過所有的頂點一次
- 漢米爾頓迴路 (Hamiltonian Cycle)
  - 經過所有頂點後回到起點的漢米爾頓路徑
  - 除起點與終點外，其餘頂點只經過一次
- 漢米爾頓圖 (Hamiltonian Graph)
  - 包含一個漢米爾頓迴路的圖
- 漢米爾頓路徑問題 (Hamiltonian Path Problem)
  - 決定漢米爾頓路徑與迴路是否存在於該圖中
  - NP-complete.



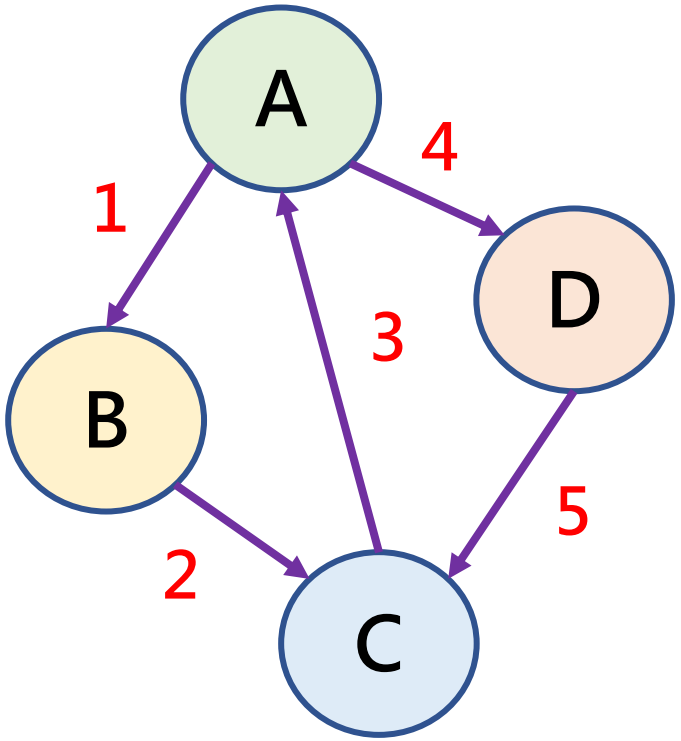
Hamiltonian Path



Hamiltonian Cycle

Euler Path and Cycle

- 尤拉路徑 (Euler Path)
  - 恰巧經過所有的邊一次
  - 起點與終點不同
  - 又稱一筆畫問題
- 尤拉迴路 (Euler Circuit/Cycle)
  - 恰巧經過所有的邊一次
  - 起點與終點相同



Euler Cycle v.s. Hamiltonian Cycle

- 漢米爾頓迴路 (Hamiltonian Cycle)
  - 經過所有頂點一次
  - 邊可重複使用
- 尤拉迴路 (Euler Circuit/Cycle)
  - 經過所有邊一次
  - 頂點可重複經過

實戰練習

Practice

Mission

Try LeetCode #1615. Maximal Network Rank

There is an infrastructure of n cities with some number of roads connecting these cities. Each roads[i] = [ai, bi] indicates that there is a bidirectional road between cities ai and bi.

The network rank of two different cities is defined as the total number of directly connected roads to either city. If a road is directly connected to both cities, it is only counted once.

The maximal network rank of the infrastructure is the maximum network rank of all pairs of different cities.

Given the integer n and the array roads, return the maximal network rank of the entire infrastructure.

Ref：https://leetcode.com/problems/maximal-network-rank/

Practice

Mission

Try LeetCode #841. Keys and Rooms

There are N rooms and you start in room 0. Each room has a distinct number in 0, 1, 2, ..., N-1, and each room may have some keys to access the next room.

Formally, each room i has a list of keys rooms[i], and each key rooms[i][j] is an integer in [0, 1, ..., N-1] where N = rooms.length. A key rooms[i][j] = v opens the room with number v.

Initially, all the rooms start locked (except for room 0).

You can walk back and forth between rooms freely. Return true if and only if you can enter every room.

Ref：https://leetcode.com/problems/keys-and-rooms/