# 平面圖

日月卦長

# 平面圖

- 可以畫在平面上且邊不相交的圖

$K_4$　　Planar Embedding of $K_4$

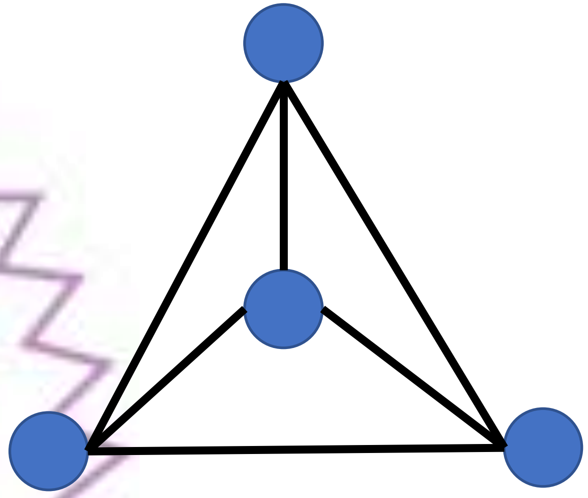
# 歐拉公式

- $V$: 點數
- $E$: 邊數
- $F$: 面數
- $C$: 連通塊數

$V = 4$
$E = 6$
$F = \mathbf{4}$
$C = 1$

$$4 - 6 + 4 = 1 + 1$$

$$V - E + F = C + 1$$



# 歐拉公式

$$V - E + F = C + 1$$

- 因為每個面都由至少 3 個邊圍成，且每個邊僅觸及 2 個面：

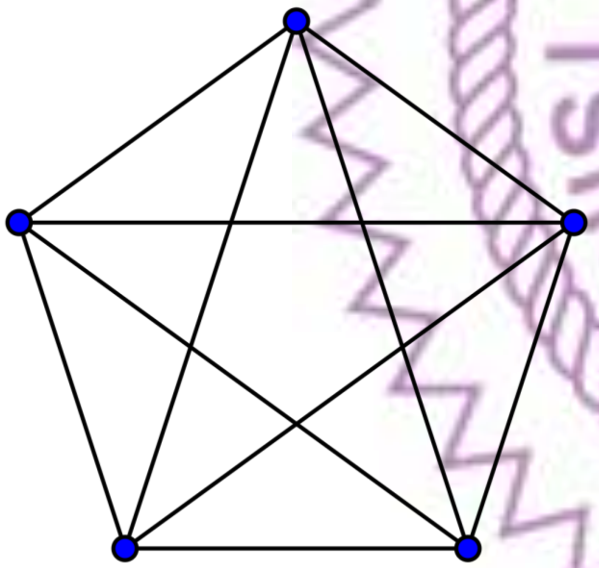
$$3F \leq 2E$$

- 因此若 $G$ 是一個連通簡單圖：

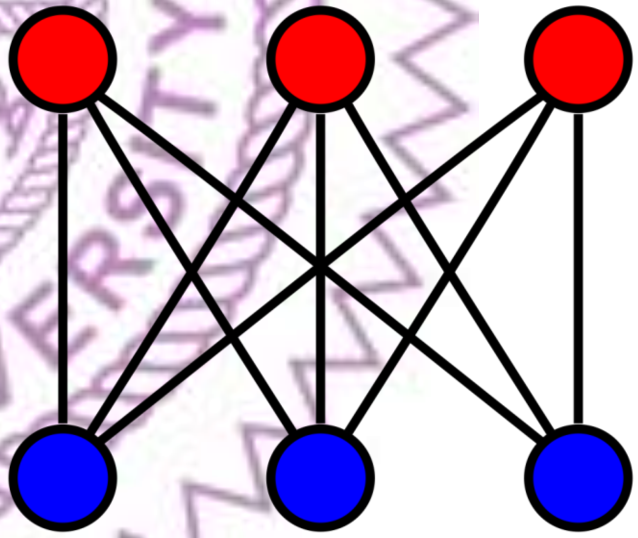
$$E \leq 3V - 6$$

# Kuratowski's theorem

- 簡單圖 $G$ 是平面圖 $\leftrightarrow$ $G$ 不包含一個子圖是 $K_5$ 或 $K_{3,3}$ 的同胚



$K_5$



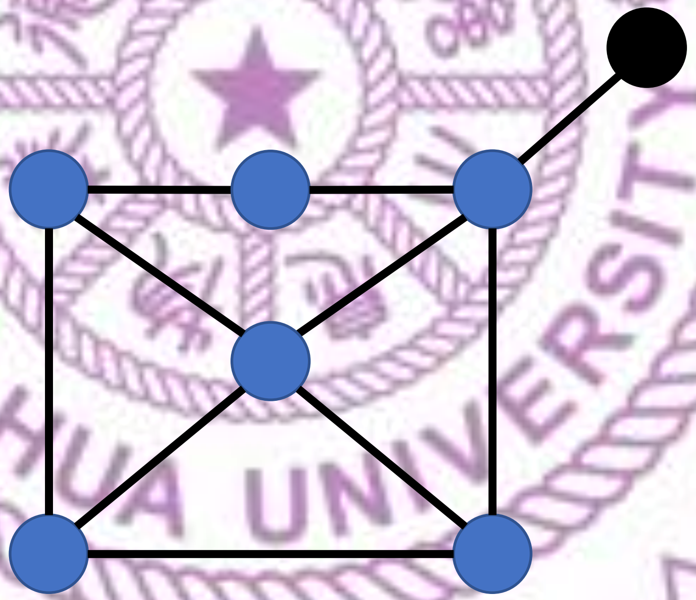
$K_{3,3}$

# 同胚 (Homeomorphism)

- **細分變換**：在一條邊上面加入新的點
- **同胚**：兩張圖在經過一些**細分變換**後會**同構**

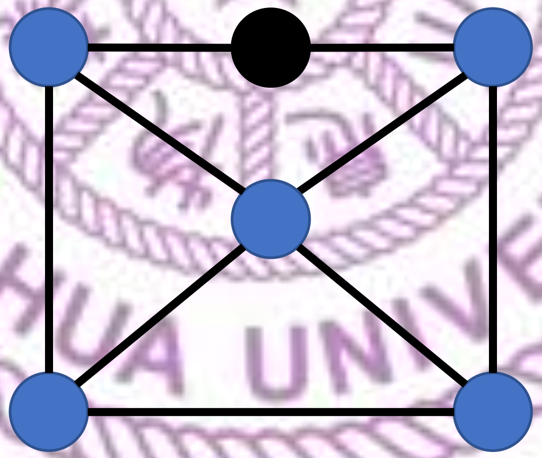
# Smoothing

- 細分變換的逆轉換
- 不斷把 $degree \le 2$ 的點移除並合併邊



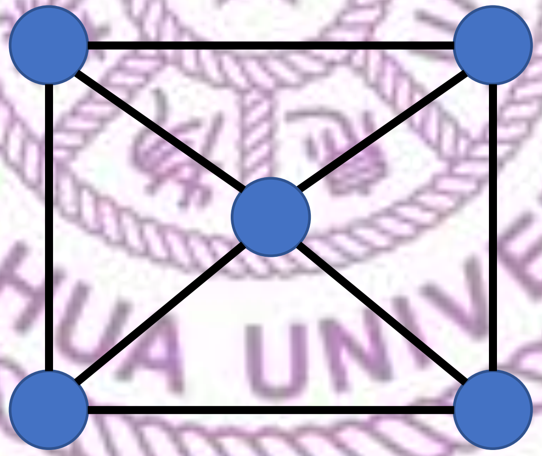
# Smoothing

- 細分變換的逆轉換
- 不斷把 $degree \leq 2$ 的點移除並合併邊



# Smoothing

- 細分變換的逆轉換
- 不斷把 $degree \leq 2$ 的點移除並合併邊



# Smoothing

- 用 adjacency matrix
- 一次移除一個點
- 時間複雜度是 $O(n^3)$

```
using AdjacencyMatrixTy = vector<vector<bool>>;
AdjacencyMatrixTy smoothing(AdjacencyMatrixTy G) {
  size_t N = G.size(), Change = 0;
  do {
    Change = 0;
    for (size_t u = 0; u < N; ++u) {
      vector<size_t> E;
      for (size_t v = 0; v < N && E.size() < 3; ++v)
        if (G[u][v] && u != v) E.emplace_back(v);
      if (E.size() == 1 || E.size() == 2) {
        ++Change;
        for (auto v : E) G[u][v] = G[v][u] = false;
      }
      if (E.size() == 2) {
        auto [a, b] = make_pair(E[0], E[1]);
        G[a][b] = G[b][a] = true;
      }
    }
  } while (Change);
  return G;
}
```

# 判斷 $K_5$ 或 $K_{3,3}$

計算 degree

```
vector<size_t> getDegree(const AdjacencyMatrixTy &G) {
  size_t N = G.size();
  vector<size_t> Degree(N);
  for (size_t u = 0; u < N; ++u)
    for (size_t v = u + 1; v < N; ++v) {
      if (!G[u][v]) continue;
      ++Degree[u], ++Degree[v];
    }
  return Degree;
}
```

```
bool is_K5_or_K33(const vector<size_t> &Degree) {
  unordered_map<size_t, size_t> Num;
  for (auto Val : Degree) ++Num[Val];
  size_t N = Degree.size();
  bool isK5 = Num[4] == 5 && Num[4] + Num[0] == N;
  bool isK33 = Num[3] == 6 && Num[3] + Num[0] == N;
  return isK5 || isK33;
}
```

透過 degree 判斷

# 平面圖判斷法

- 枚舉子圖，對個子圖做 Smoothing 後判斷是否是 $K_5$ 或 $K_{3,3}$
- 複雜的 O(n) 演算法
  [A new planarity test](#)

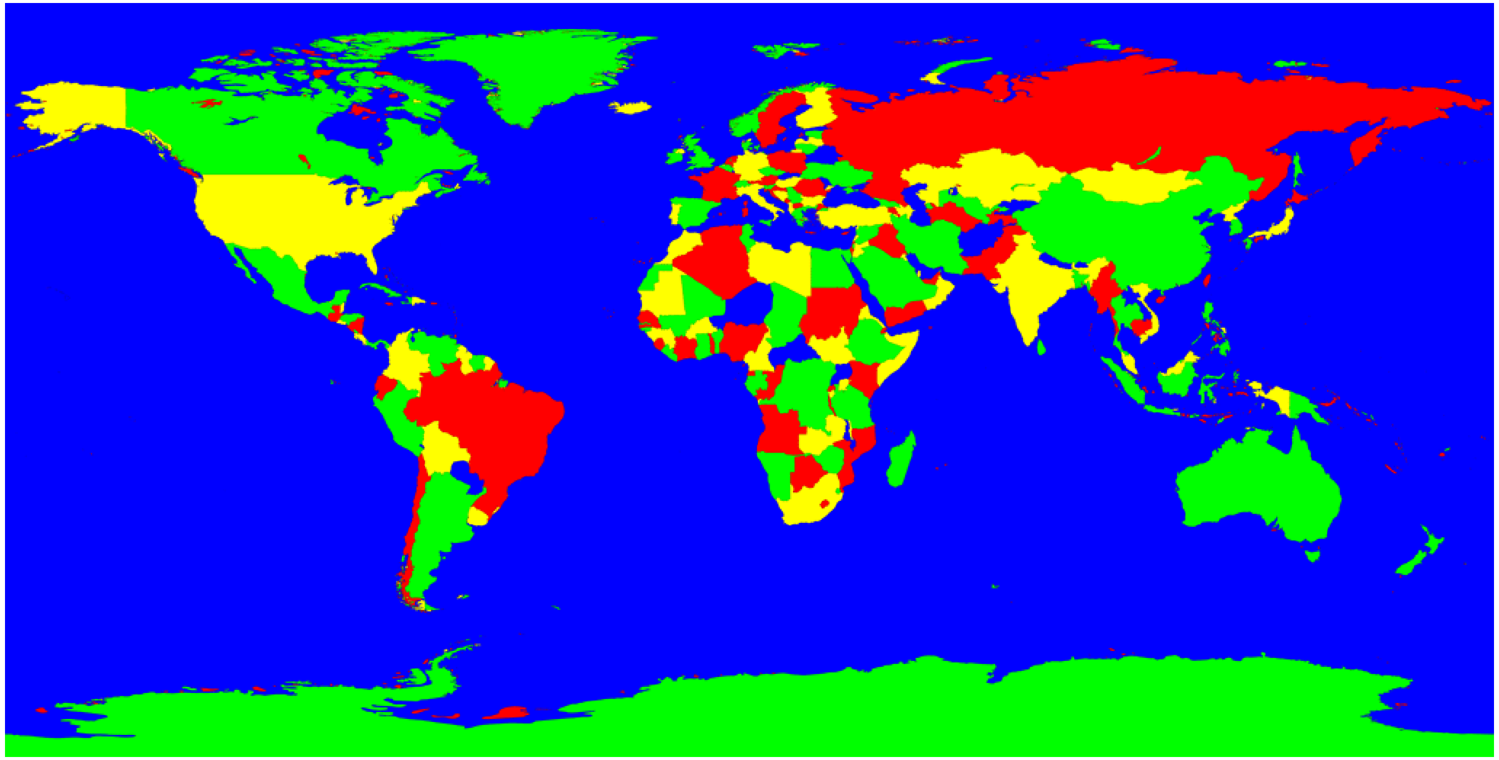
# 圖色數 (圖染色問題)

- 將一張圖上的每個頂點染色，使得相鄰的兩個點顏色不同，最小需要的顏色數。

- 通常使用符號 $\chi(G)$ 表示

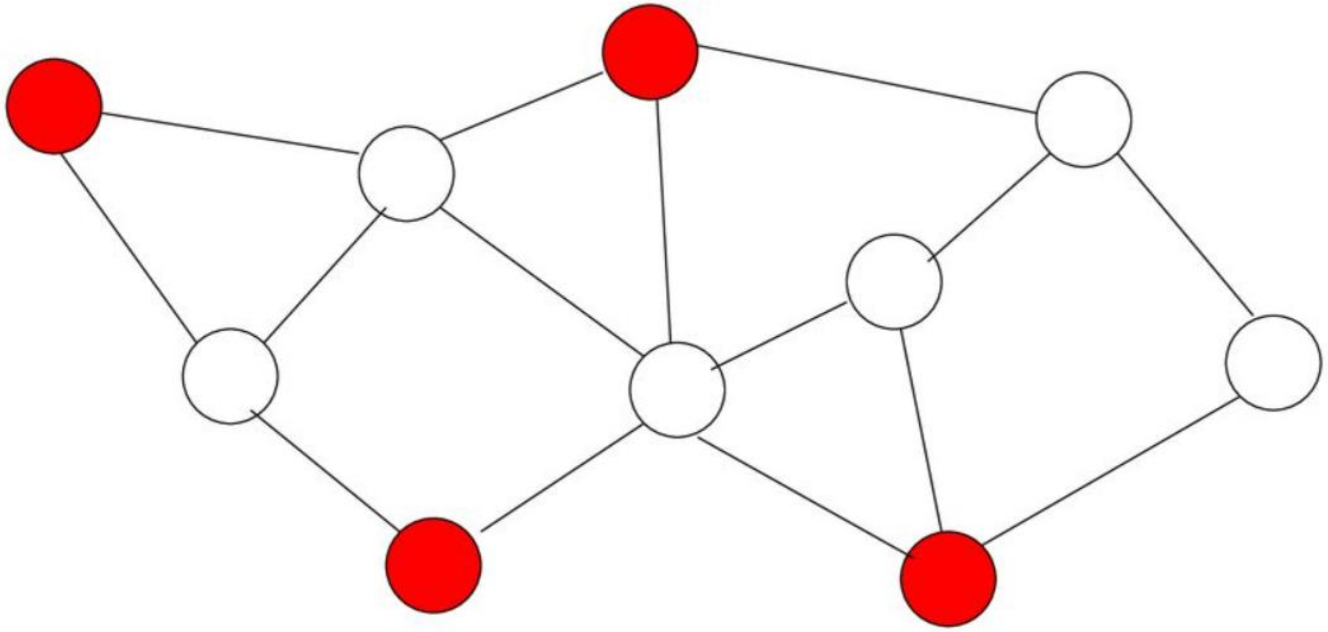
# 四色定理

若圖 G 是平面圖，則 $\chi(G) \leq 4$ 。



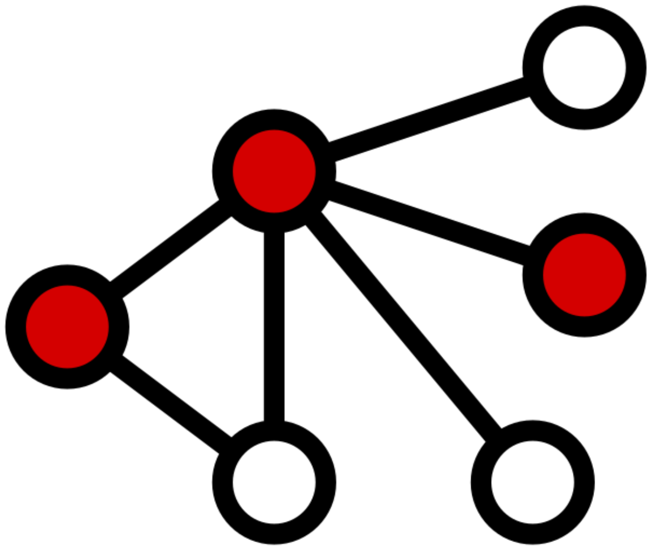
# 判斷平面圖的圖色數

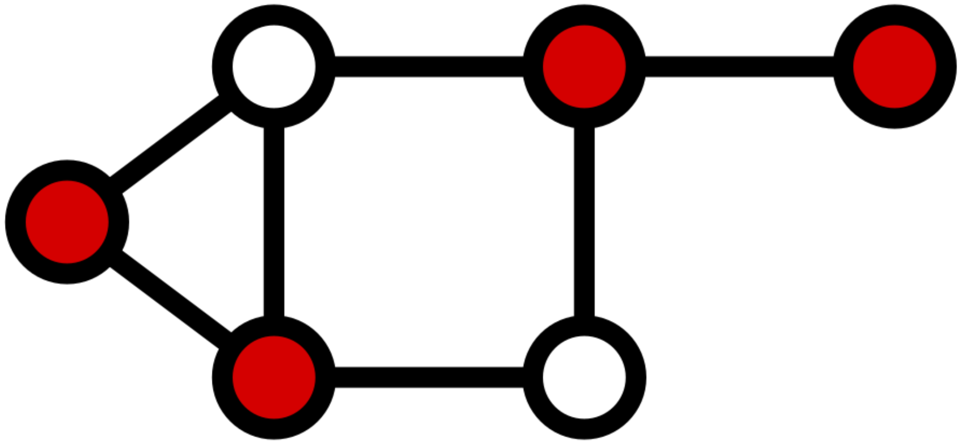
- 判斷 2 塗色 ➔ 等同於判斷二分圖
- 判斷 3 塗色 ➔ NP-Complete
- 判斷 4 塗色 ➔ 有多項式時間演算法，但我不會



# 獨立集

一張圖選一些點，使得他們之間沒有邊相連



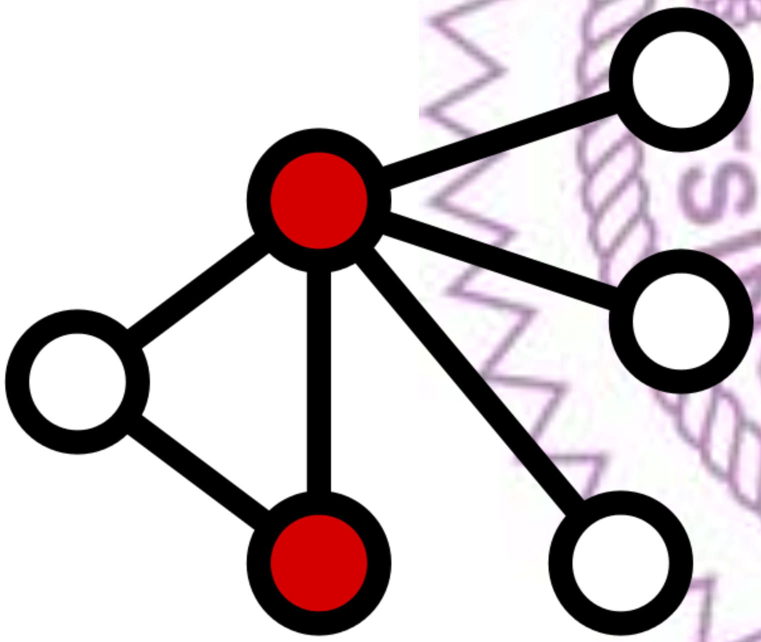


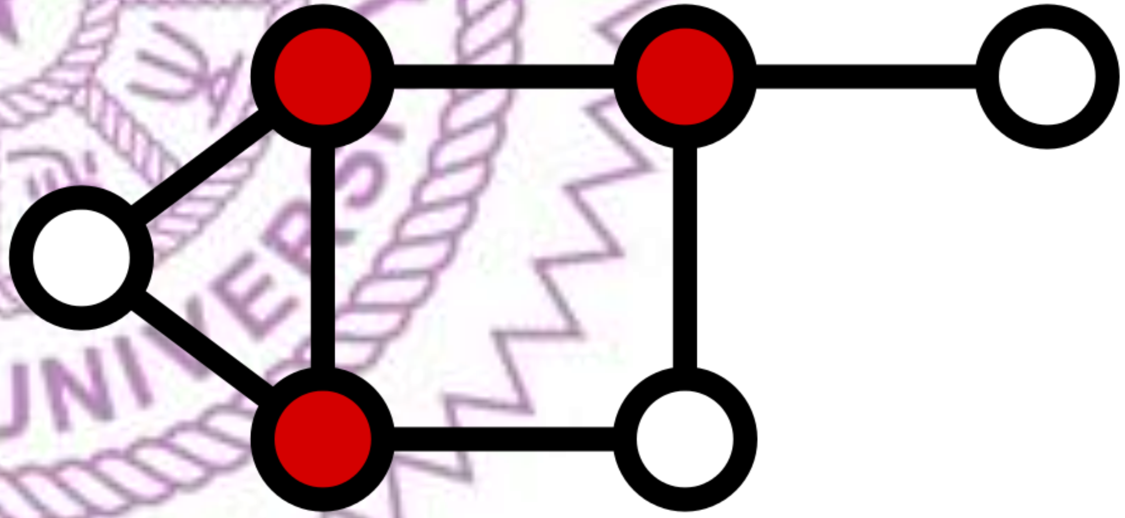
# 頂點覆蓋

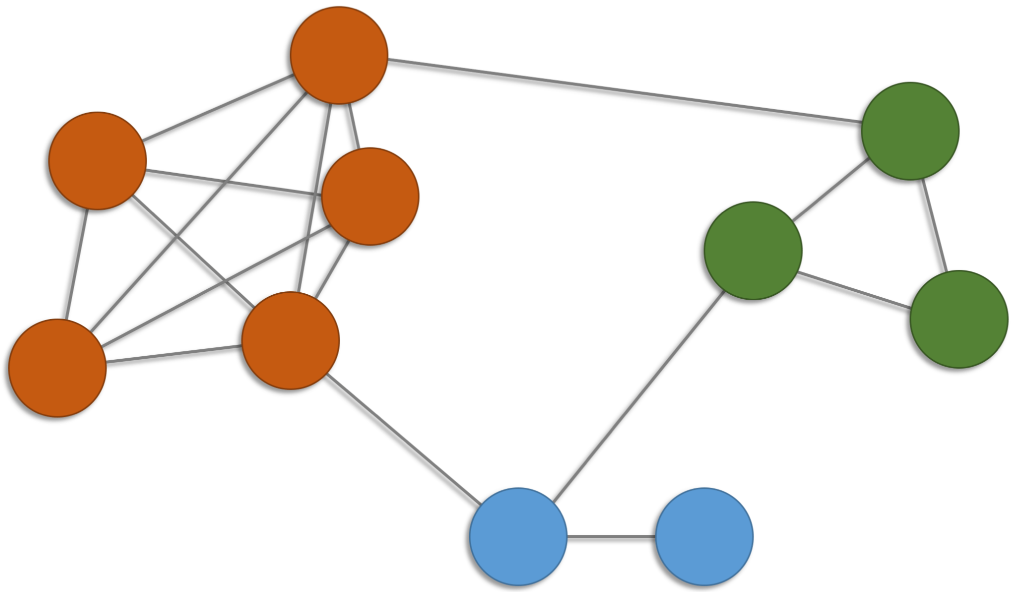
- 設 $G = (V, E)$
- 一個頂點集合 $V' \subset V$
  使得 $\forall (u, v) \in E, u \in V' \vee v \in V'$

# 每個頂點覆蓋的補集都對應一個獨立集

最大獨立集點數 + 最小頂點覆蓋點數 = $|V|$





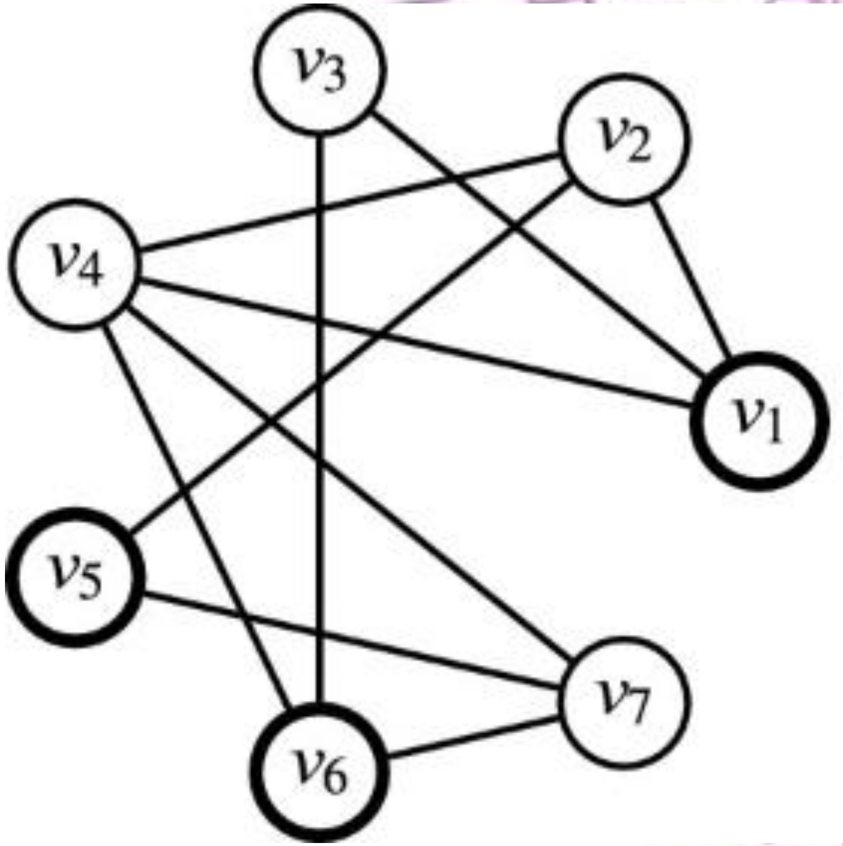


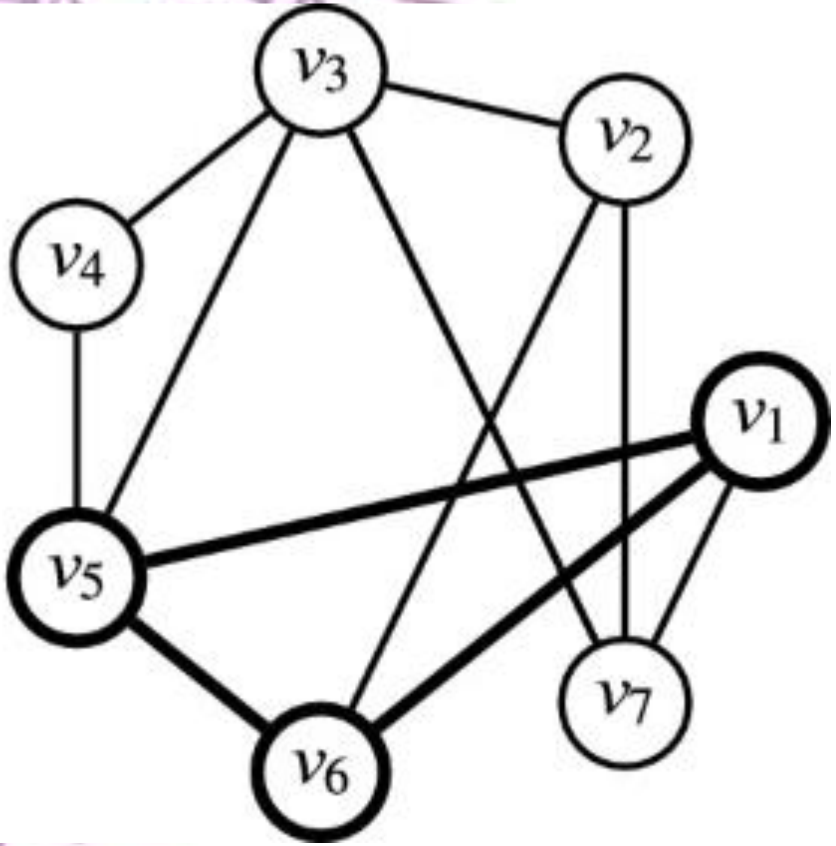
# 團

無向圖中，滿足兩兩之間有邊連接的頂點的集合

# 每個獨立集在補圖中都對應一個團

最大獨立集 = 補圖的最大團



# 枚舉極大團 (Bron–Kerbosch algorithm)

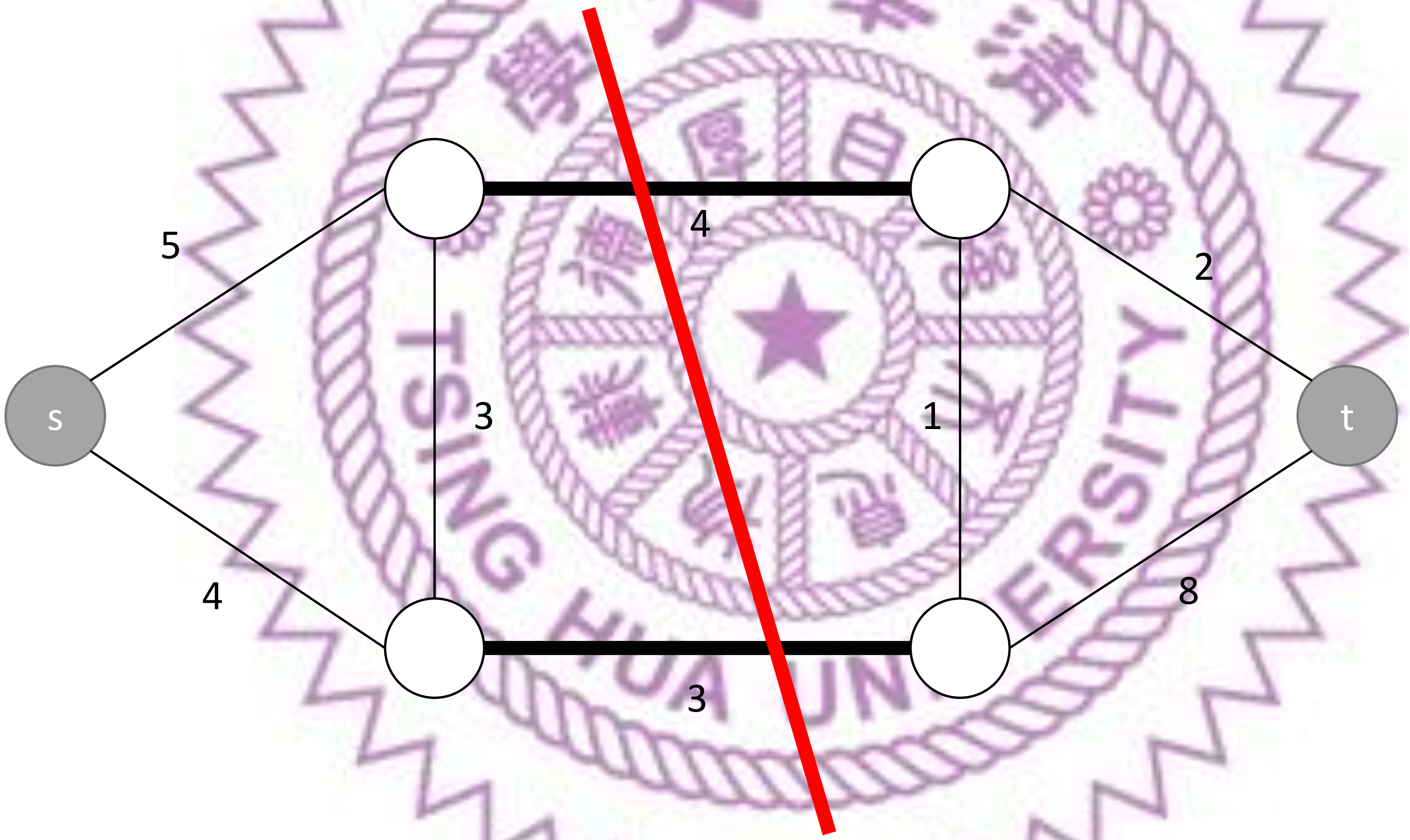
- 極大團：增加任一頂點都不再符合團定義的團

- $n$ 個點的圖最多有 $3^{n/3}$ 個極大團
- Bron–Kerbosch algorithm 可以枚舉所有極大團，時間為 $O(3^{n/3})$
- 請將這個演算法加入模板

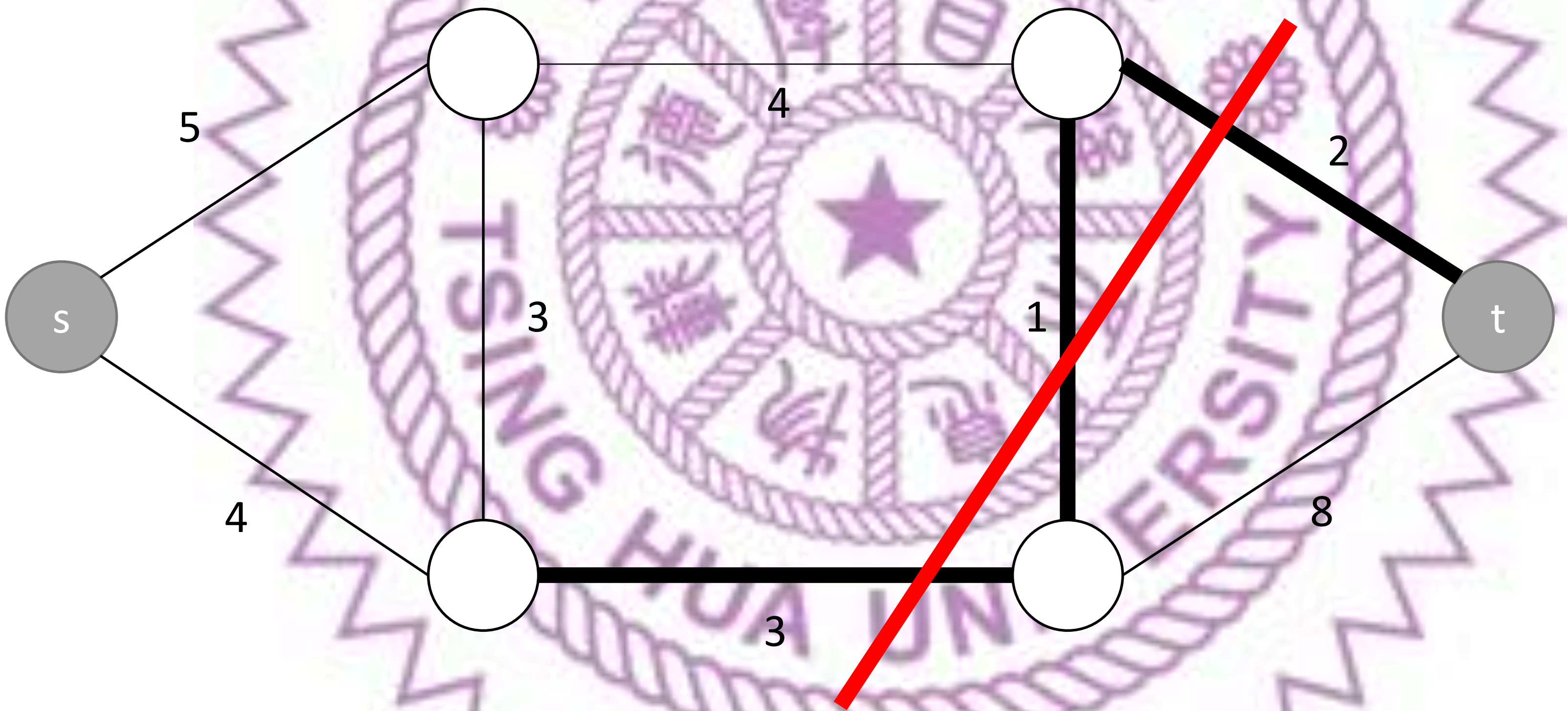
# 判斷圖 $G$ 是否能 3 塗色

- 枚舉圖 $G$ 的極大獨立集 $I$
- 若存在 $I$ 使得 $G - I$ 形成二分圖，則 $G$ 可以 3 塗色
- 反之則不能 3 塗色

# $s,t-$割 $(s,t-cut)$



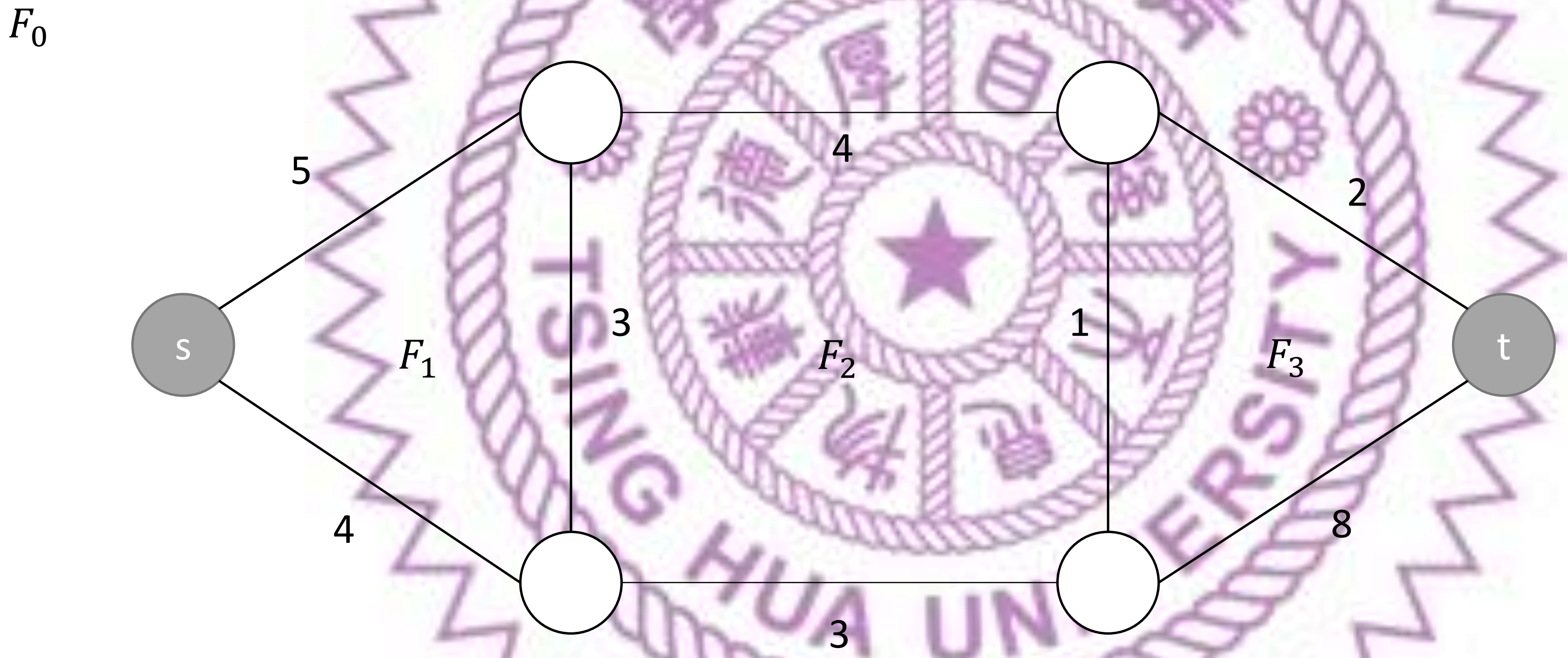
# $s, t -$最小割 $(s, t - mincut)$



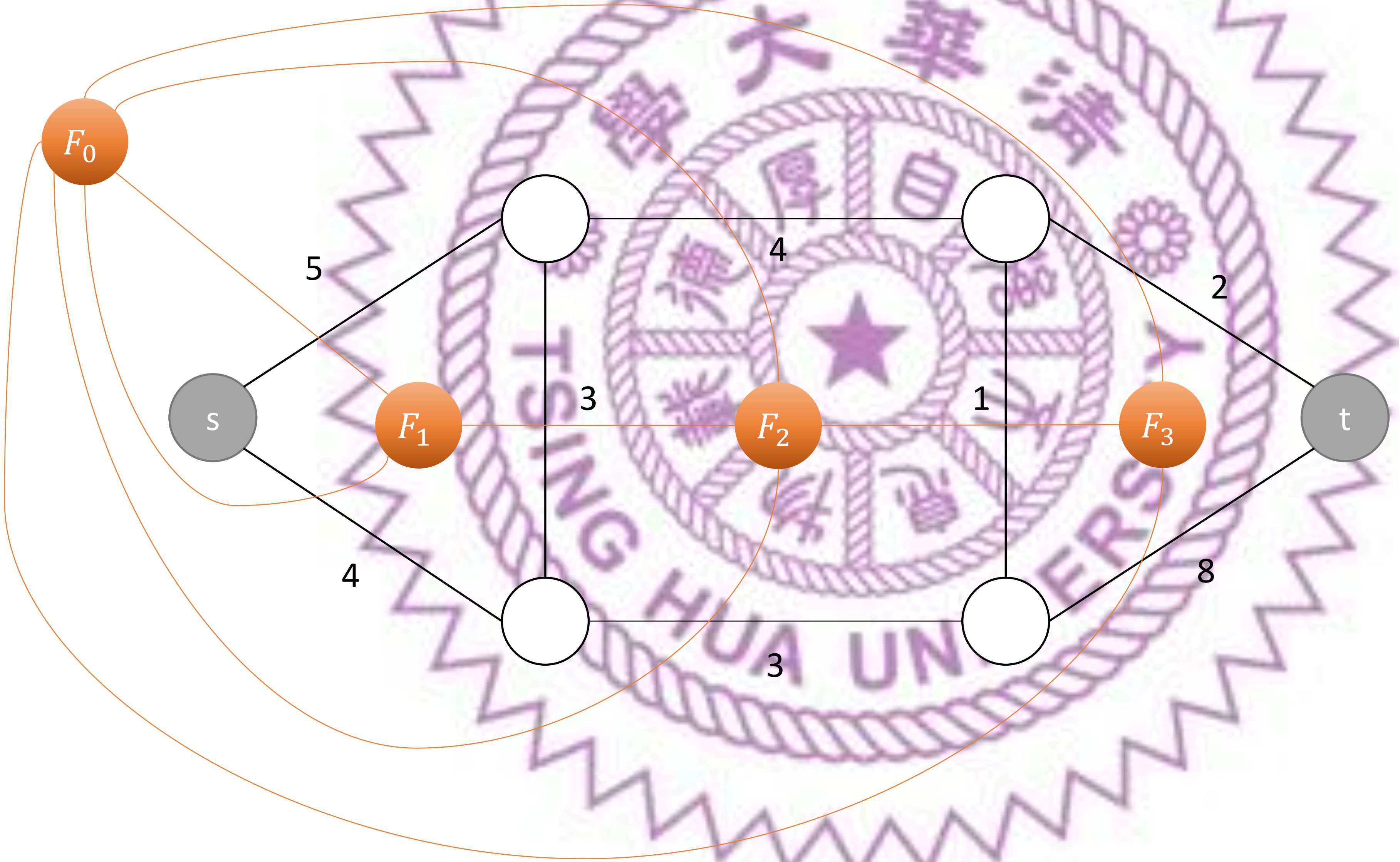
# 平面圖 $s, t$ −最小割

- Ford and D. Fulkerson, “Maximal flow through a network”, 1956
- 假設圖是連通無向圖，且邊的權重都是正的
- 若 $s, t$ 兩點位於同一個面 (face) $F_0$ 上，則 $s, t - mincut$ 可以對應到**對偶圖 (Dual Graph)** 上通過 $F_0$ 的一個最小環 $(s, t - F_0 - mincycle)$
- 可以用最短路徑演算法構造

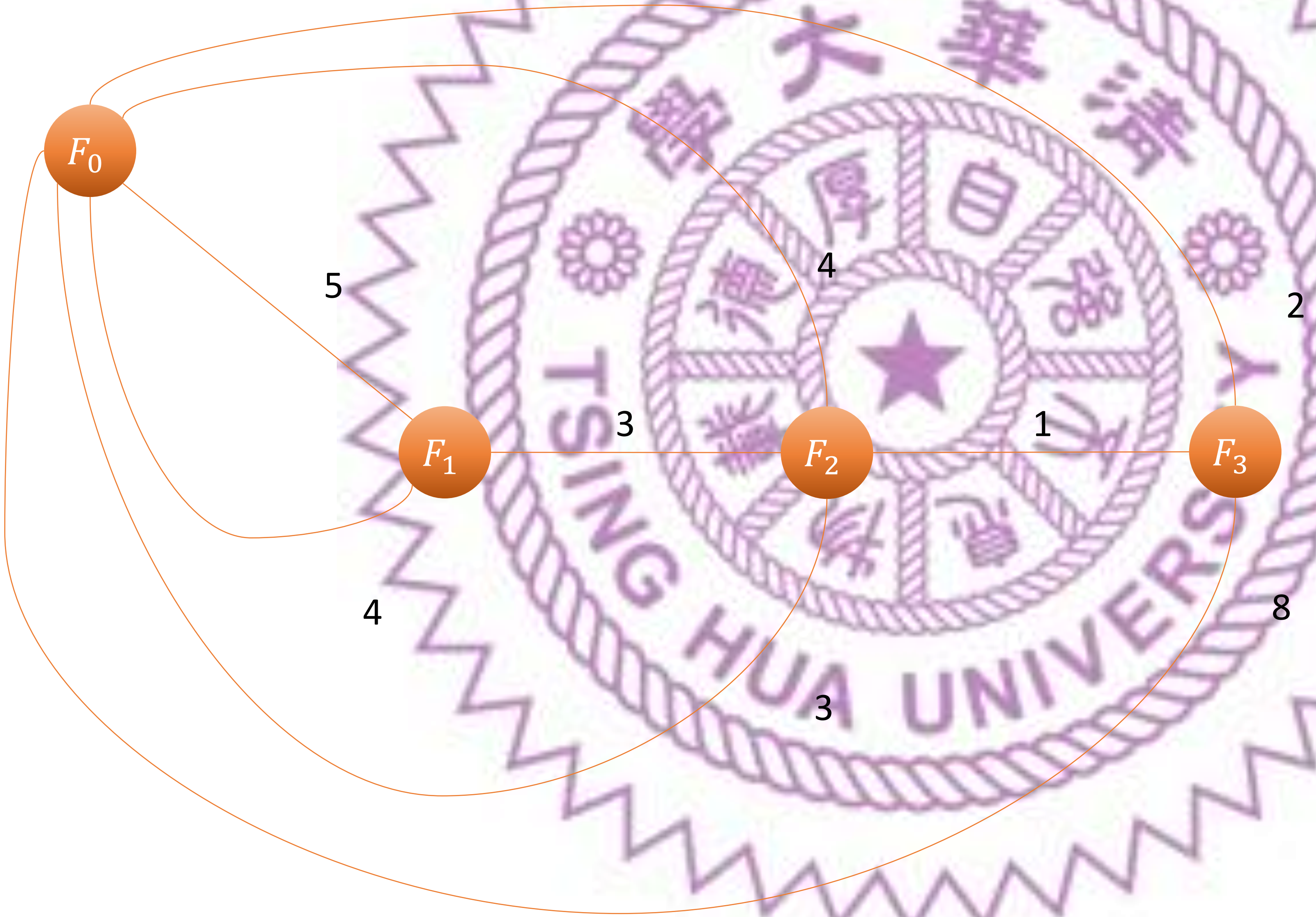
# 對偶圖 (Dual Graph)



# 對偶圖 (Dual Graph)

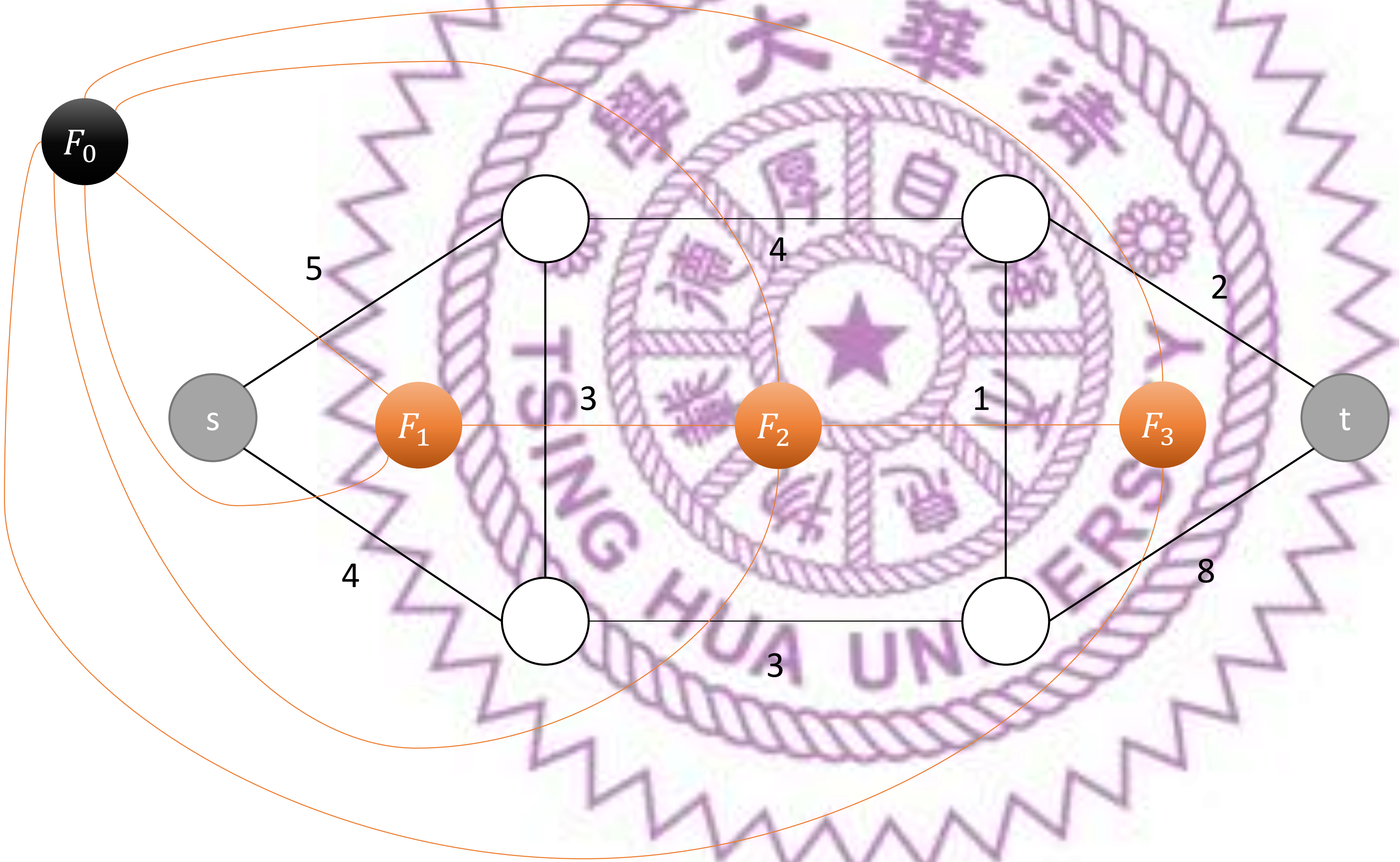


# 對偶圖 (Dual Graph)




- 原本的：
  - 點變成面
  - 面變成點
- 邊能互相對應

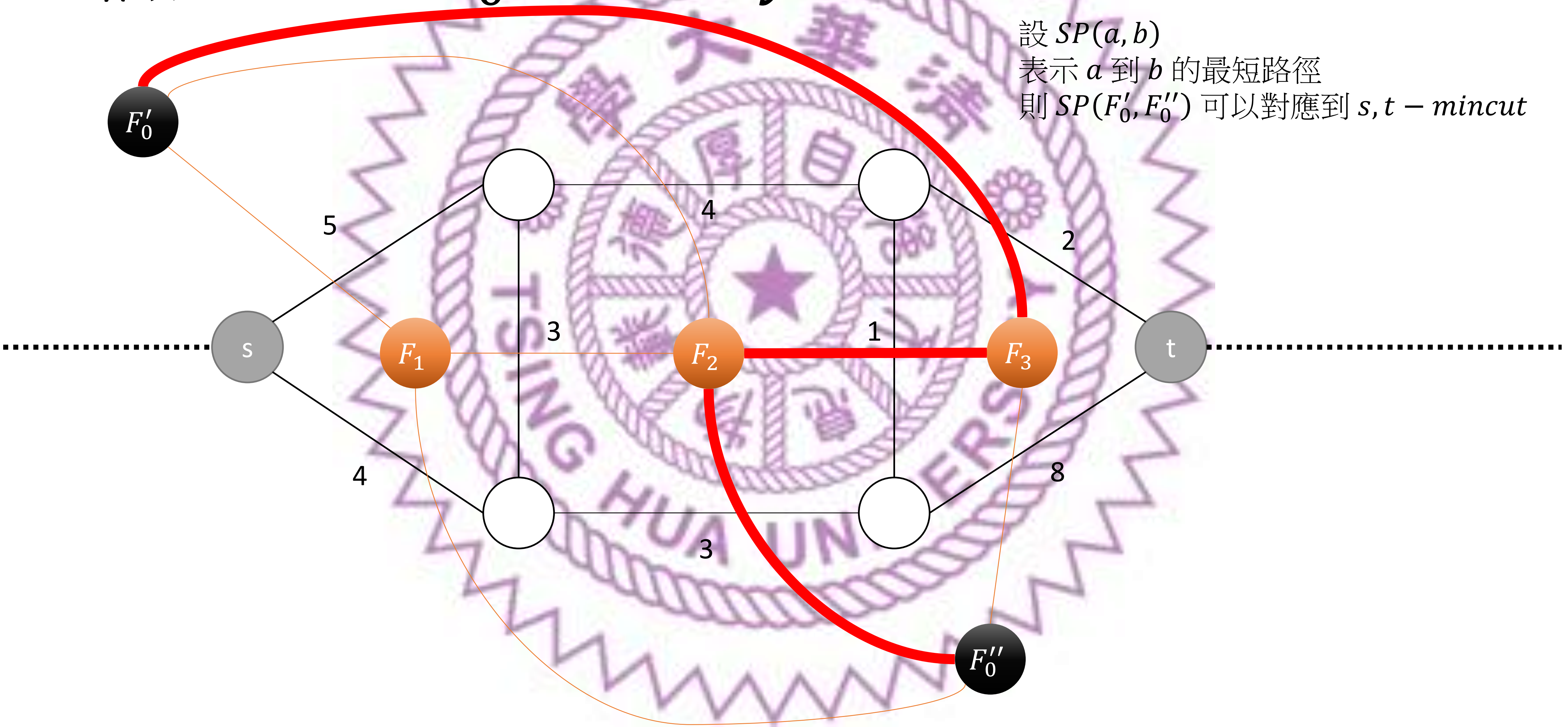
# 構造 $s,t-F_0-mincycle$



# 構造 $s, t - F_0 - mincycle$

# 構造 $s,t-F_0-mincycle$

設 $SP(a,b)$
表示 $a$ 到 $b$ 的最短路徑
則 $SP(F_0',F_0'')$ 可以對應到 $s,t-mincut$



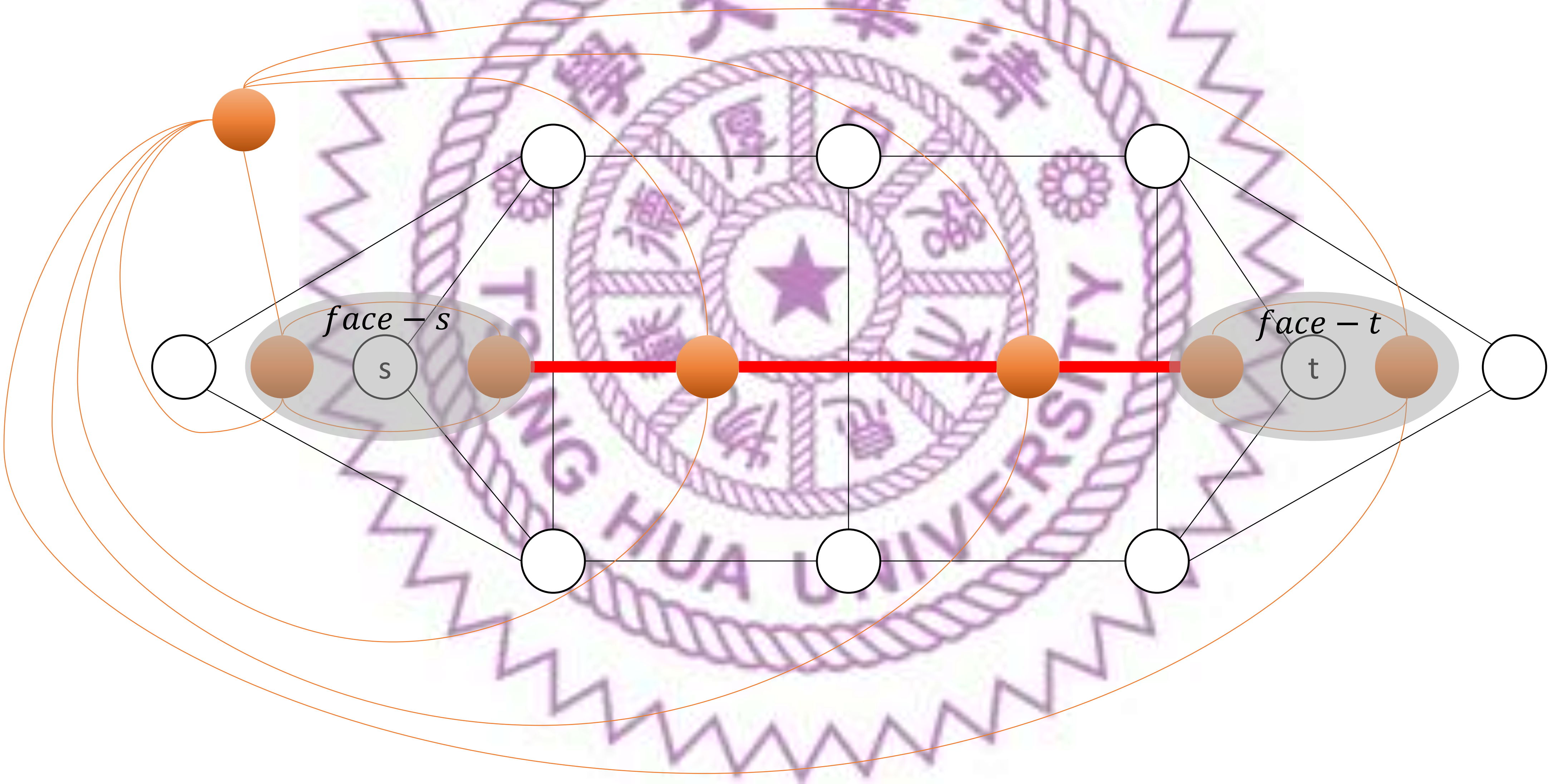
# 若 $s, t$ 不在同一個面上

- J. Reif. Minimum s-t cut of a planar undirected network in $O(n \log^2 n)$ time. 1983.
- 假設圖是連通無向圖，且邊的權重都是正的
- 利用 Ford Fulkerson 的結果做分治 (Divide and Conquer)

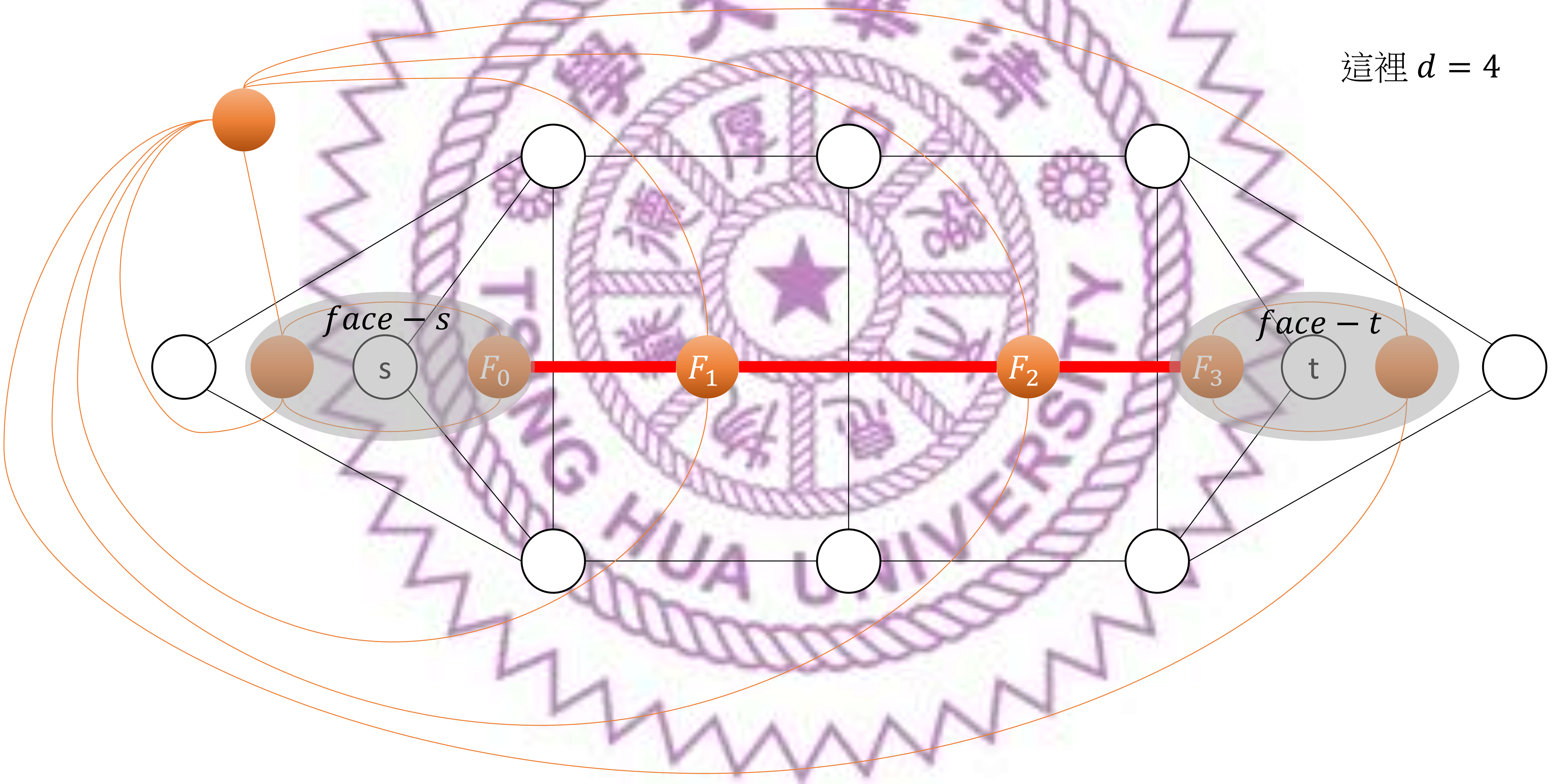
# 若 $s, t$ 不在同一個面上

假設邊有權重但太多了不想畫

# 1. 計算 $face - s$ 到 $face - t$ 的最短路徑

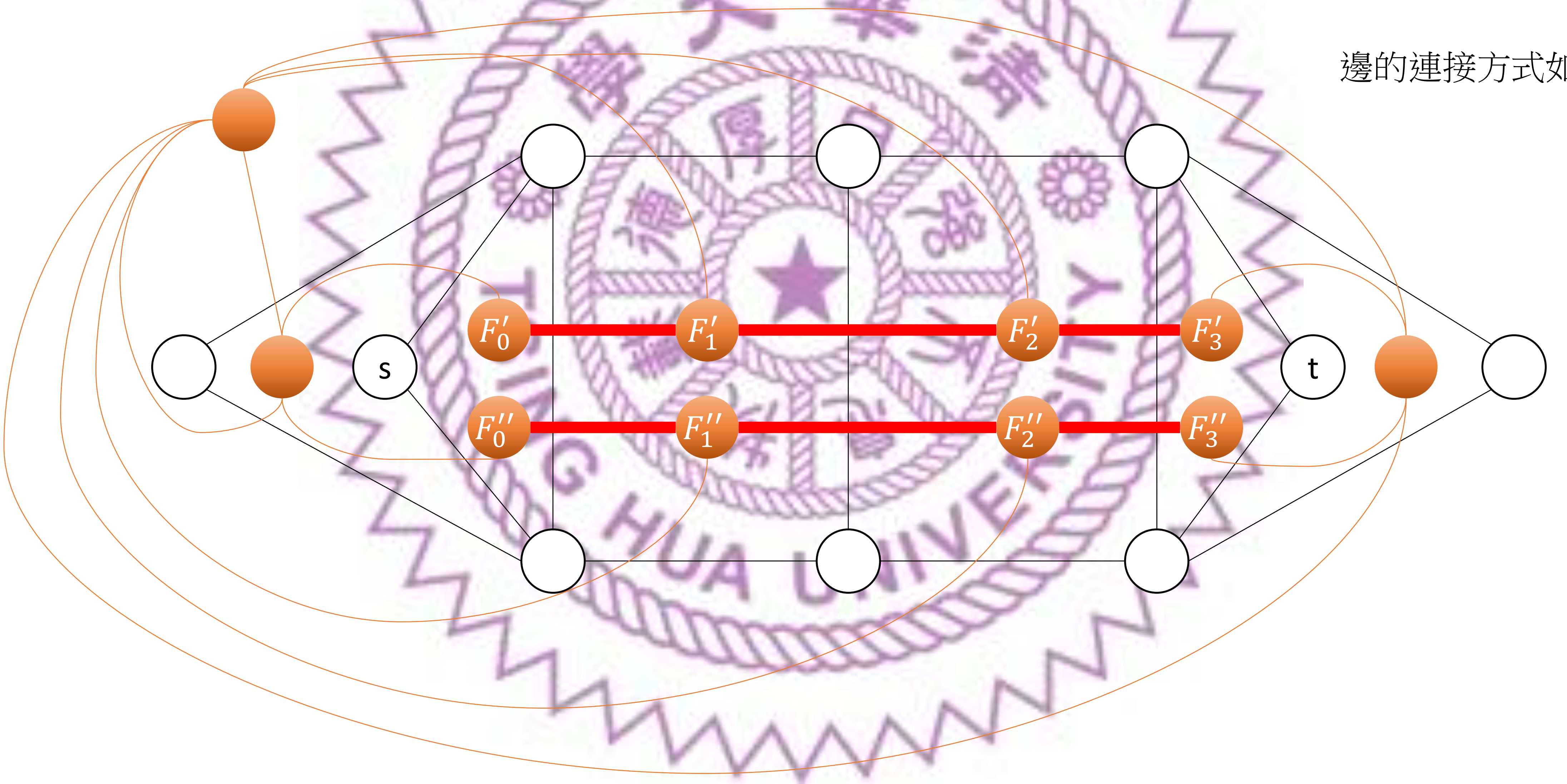


# 2. 將路過的點編號 $F_0 \sim F_d$



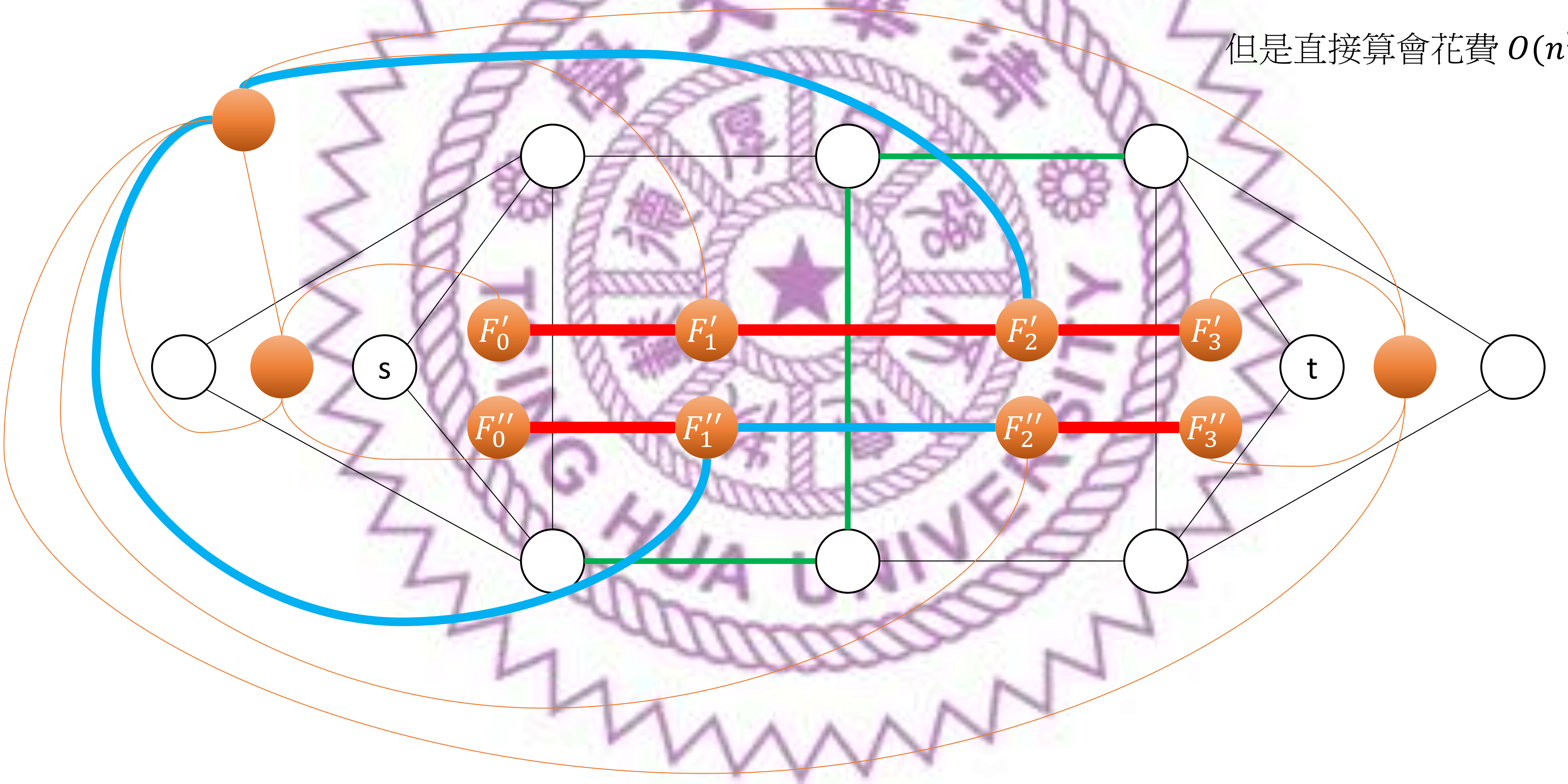
# 3. 拆點 $F_i \rightarrow F_i', F_i''$

邊的連接方式如圖所示



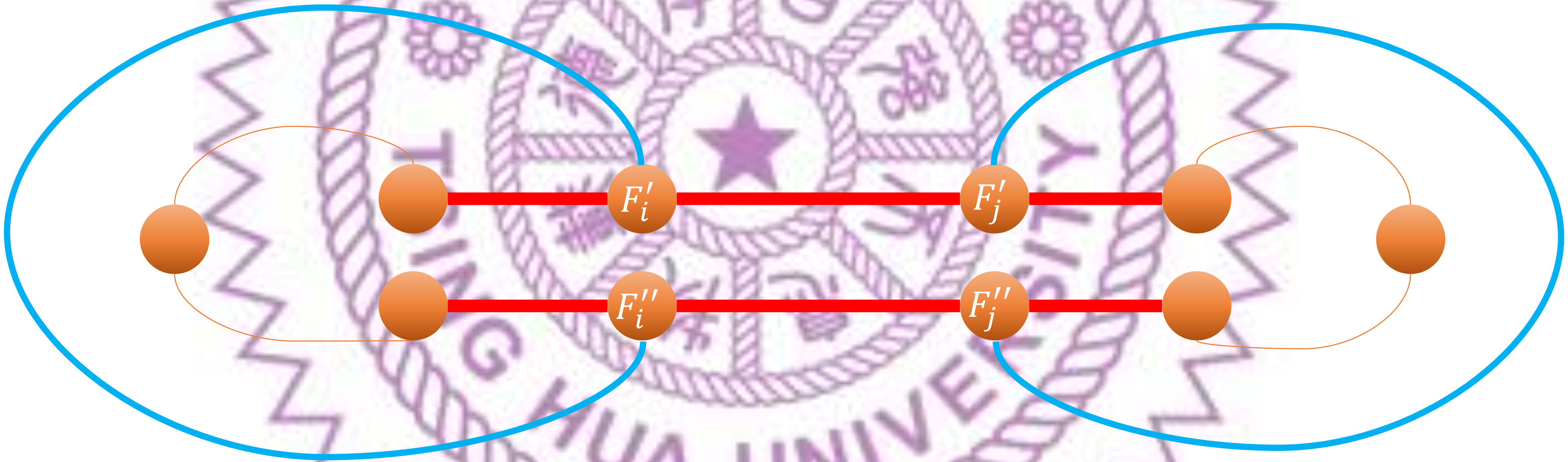
$$4.\ s,t-mincut=\min_{1\le i\le d}\{SP(F_i',F_i'')\}$$

但是直接算會花費 $O(n^2\log n)$



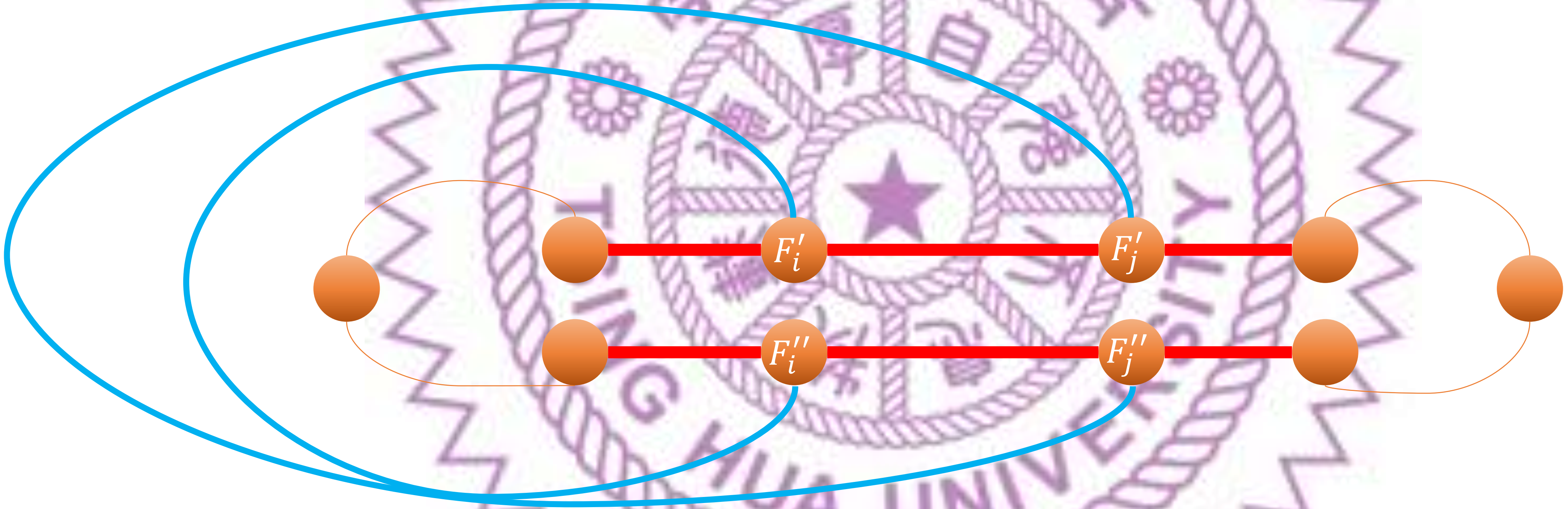
# 觀察可能的路徑，假設 $i < j$

Case 1: $SP(F_i', F_i'')$ 和 $SP(F_j', F_j'')$ 位於不同方向



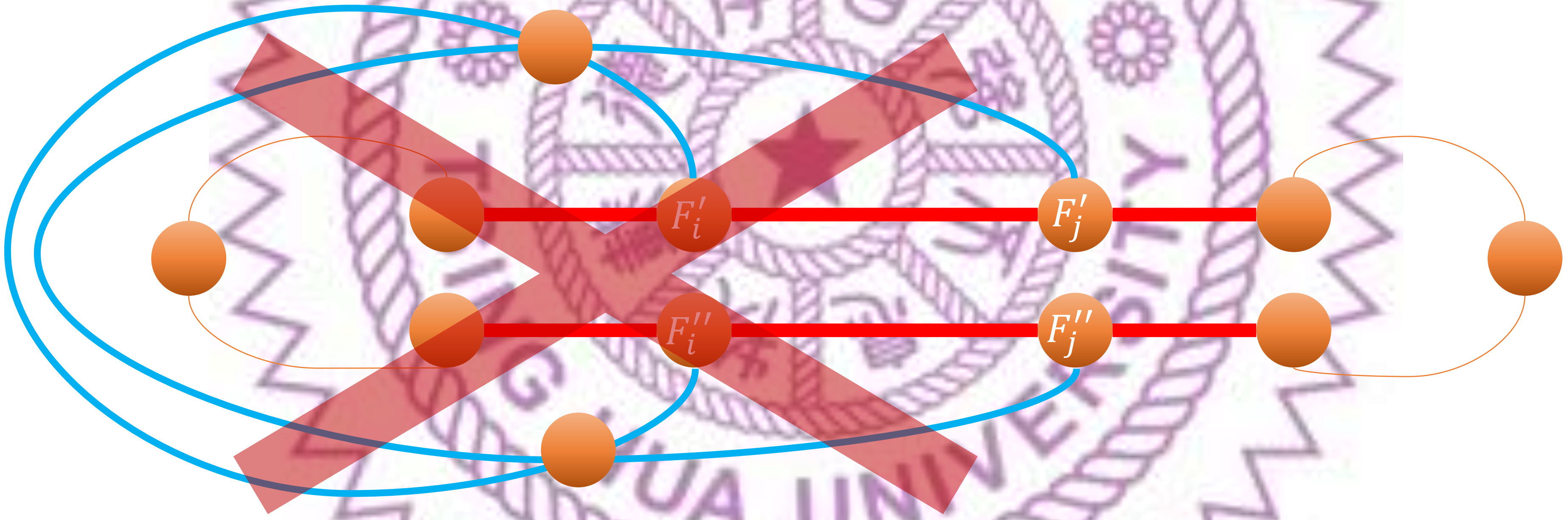
# 觀察可能的路徑，假設 $i < j$

Case 2: $SP(F_i', F_i'')$ 和 $SP(F_j', F_j'')$ 位於同方向



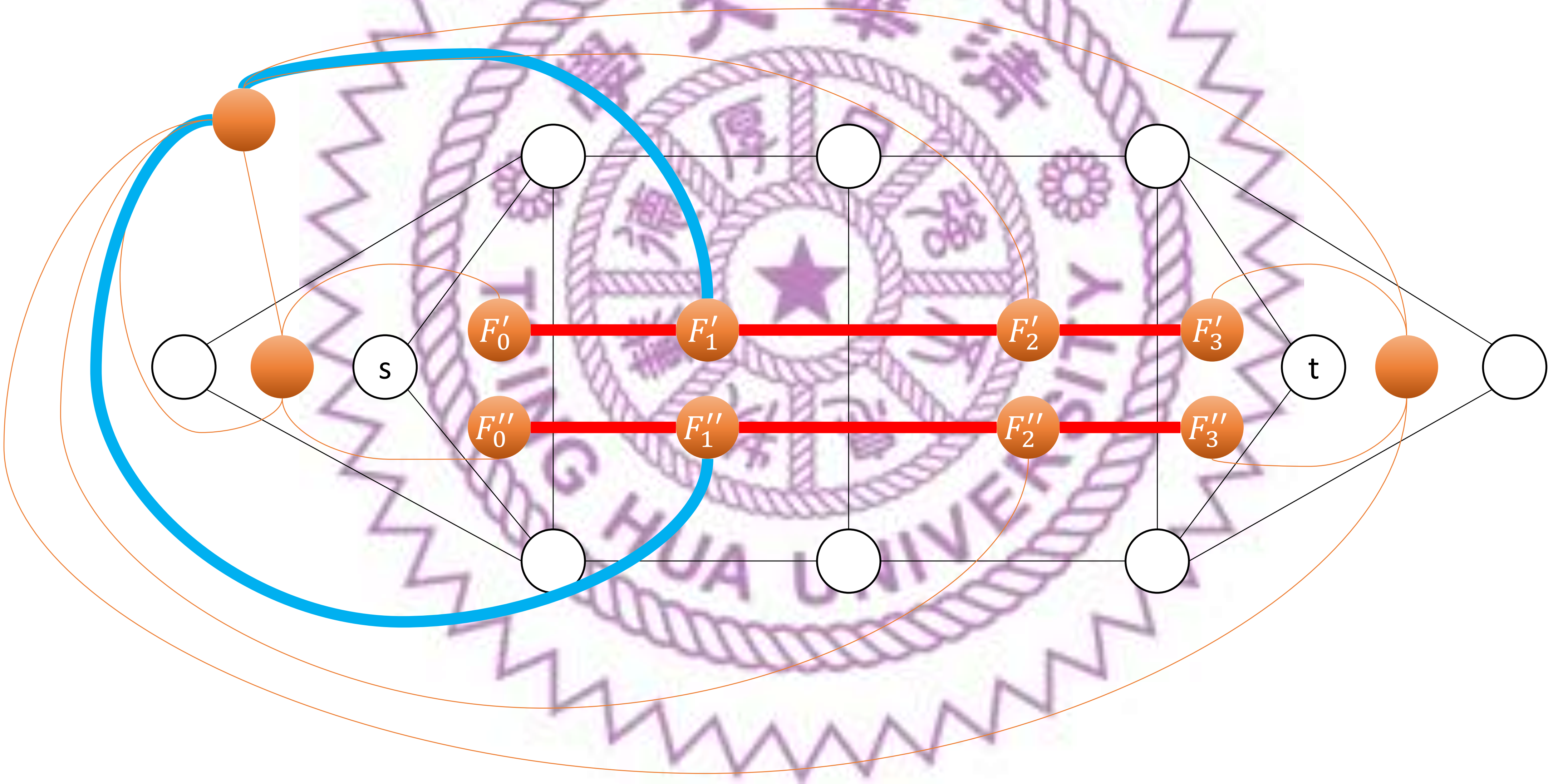
# 觀察可能的路徑，假設 $i < j$

Case 3: $SP(F_i', F_i'')$ 和 $SP(F_j', F_j'')$ 位於同方向
且沒有完全「包住」的情況

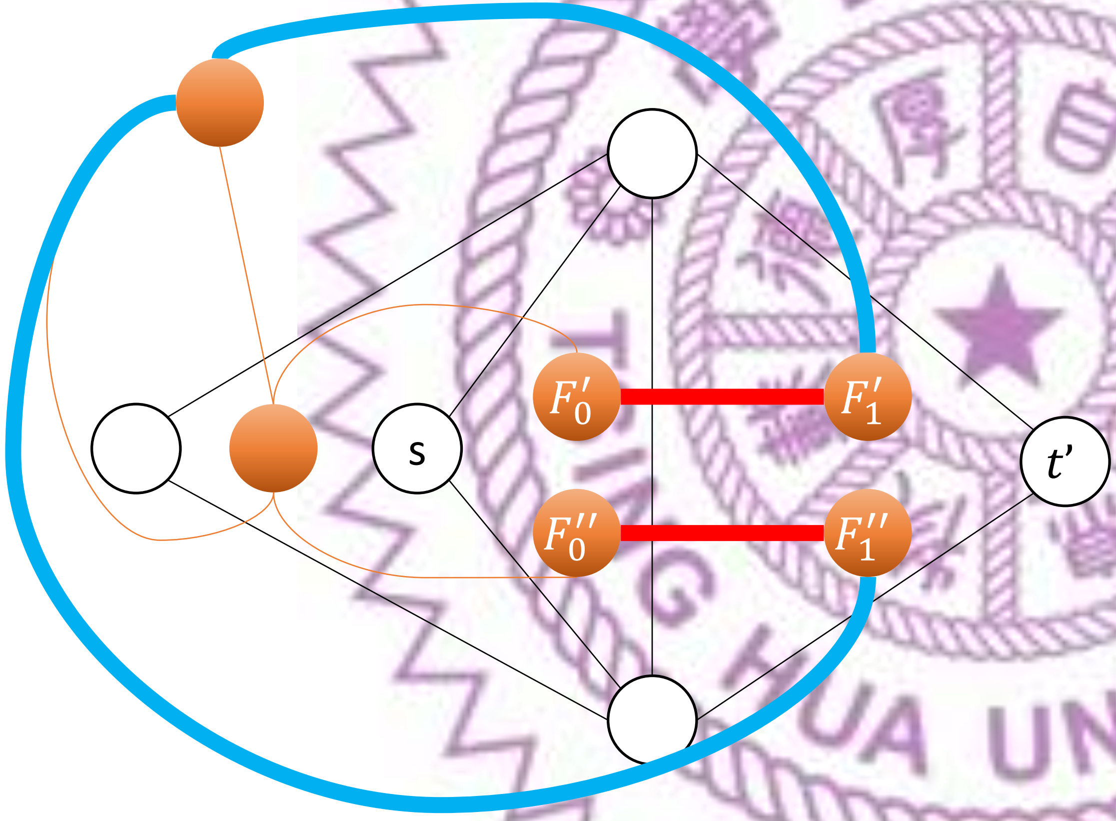



不可能存在兩條不同長度的最短路徑

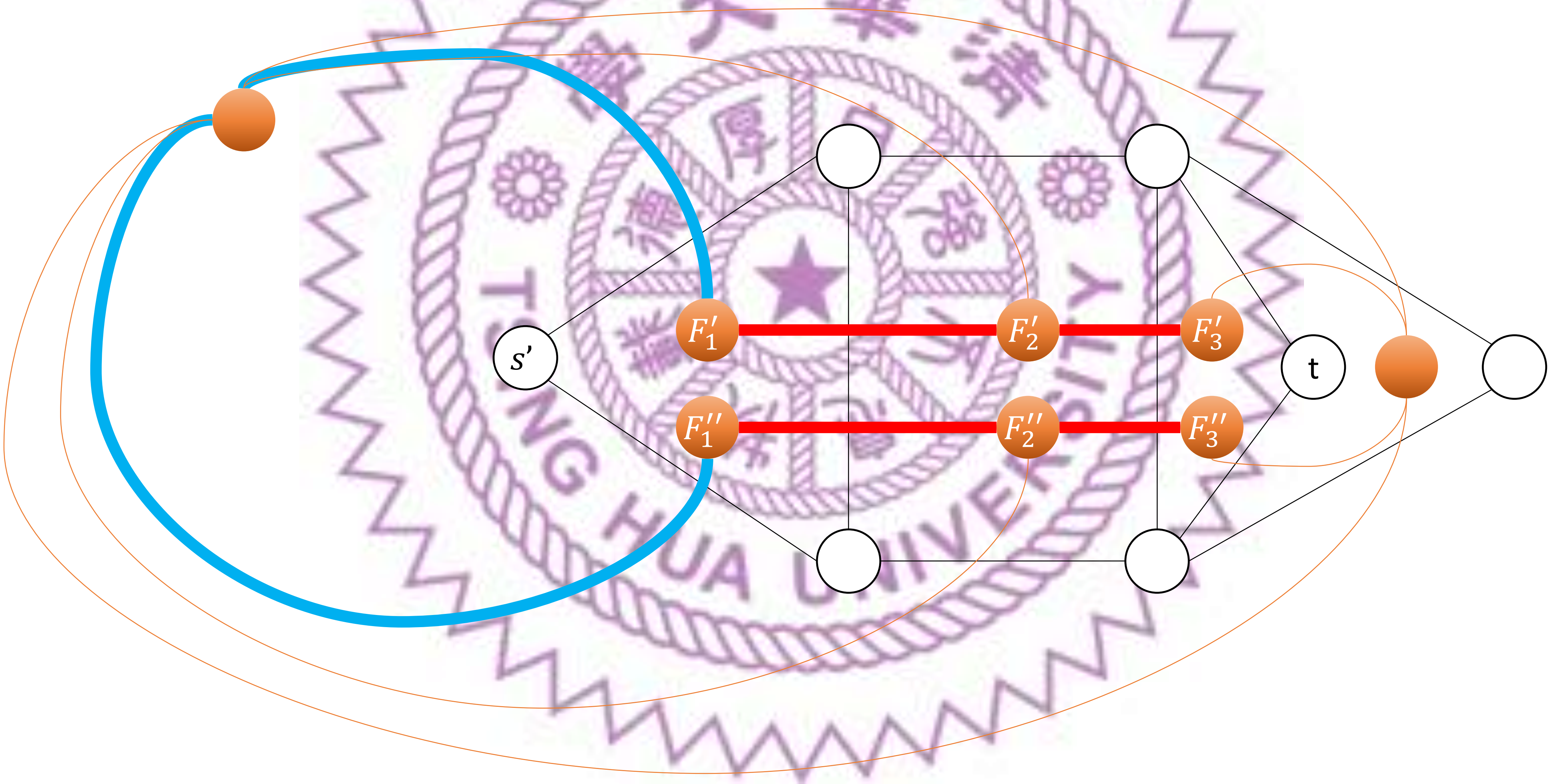
# 5. 分治法 – 先計算 $SP\left(F'_{\lfloor d/2 \rfloor}, F''_{\lfloor d/2 \rfloor}\right)$



# 5. 分治法 – 切成兩張圖遞迴處理



# 5. 分治法 – 切成兩張圖遞迴處理



# 5. 分治法－若出現 $degree = 2$ 的點要縮點

# 時間複雜度

- 遞迴結束條件：$s,t$ 位於同一個平面上就直接計算
- 遞迴深度最多 $O(\log n)$
- 每一層只有 $O(n)$ 個點
- 計算最短路徑時間： $O(n\log n)$

- 總共時間複雜度

$$O(n\log n)\times O(\log n)=O(n\log^2 n)$$