Algorithm

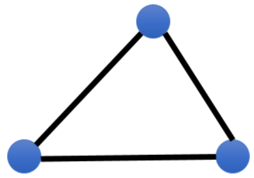
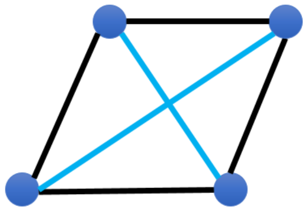
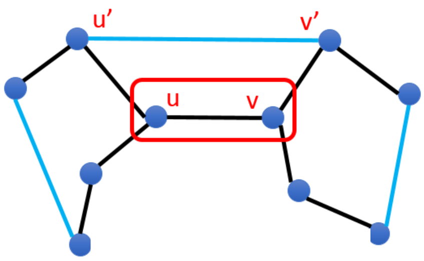
HW14

1. Let *G* be a connected, undirected graph with at least 3 vertices, and let *G3* be the graph obtained by connecting all pairs of vertices that are connected by a path in *G* of length at most 3. Prove that *G3* is hamiltonian. (Hint: Construct a spanning tree for *G*, and use an inductive argument.)

    歸納法：

    ① Prove any spanning tree of N >=3 has a Hamiltonlian cycle

    ② Prove True for all N > 4

1. u' -> v -> u => d(u', v) = 2

2. u' -> v' -> v -> u => d(u', v') = 3

3. u' -> v'-> … -> v -> u => d(u', v') = 3

2. Let 2-CNF-SAT be the set of satisfiable Boolean formula in CNF with exactly 2 literals per clause. Show that 2-CNF-SAT ∈ P. Make your algorithm as efficient as possible. (Hint: Observe that $x \vee y$ is equivalent to $\neg x \rightarrow y$. Reduce 2-CNF-SAT to an efficiently solvable problem on a directed graph.)

    2-CNF-SAT 轉為圖的問題，並證明其屬於 P

    ($x \vee y$)代表

    if not x then y

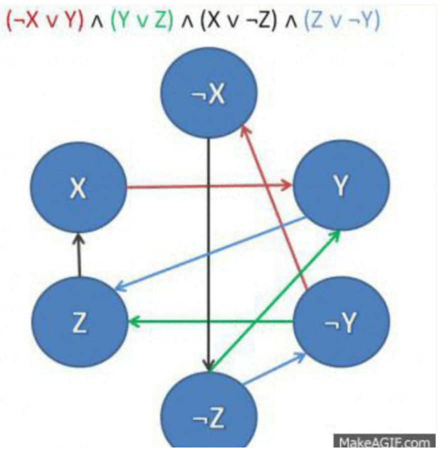
    else if NOT y then x

    if 條件形成 edge

    以下圖第一個 clause 為例

    $\neg x \vee y$ 會有兩個 edge，edge(x, y)、edge($\neg y$, $\neg x$)

    以此規則將圖建立



接著對此圖跑 DFS，檢查是否存在 path 可從¬x 到 x (對 y, z 用相同方法檢查)

如果所有變數都沒有則為 SAT，反之則非 SAT

假設有 n 個變數，則有 2n 個頂點

有 m 個 clause，每個 clause 產生 2 個邊，共有 2m 個邊

DFS 時間複雜度為 O(|V|+|E|) ，每個變數跑一次

-> $O(n*(2n + 2m)) = O(n^2)$

polynomial time -> 因此 2-CNF-SAT 為屬於 P

3. A hamiltonian path in a graph is a simple path that visits every vertex exactly once.

   a. Show that the language HAM-PATH = { (G, u, v): there is a hamiltonian path from u to v in graph G } belongs to NP.

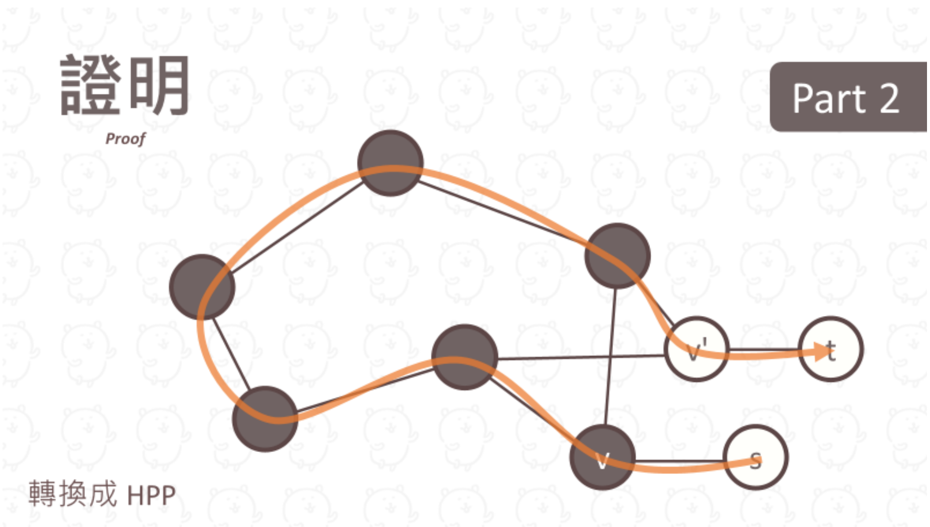
   b. Show that the Hamiltonian path problem is NP-complete. (Hint: Reduce from HAM-CYCLE.)

證明問題是 NP-complete 需證明 (i)問題是 NP
(ii)問題是 NP-hard

(i) Hamiltonian path problem 須將答案每個點走過即可，∈ NP

(ii) 已知 Hamiltonian cycle problem ∈ NP，可將 Hamiltonian cycle problem 在多項式時間轉換成 Hamiltonian path problem，將起點 V 複製，複製的點令為 V’，從 V 走到 V’即為 Hamiltonian path problem

示意圖：



4. The subgraph-isomorphism problem takes two undirected graphs G 1 and G 2 , and it asks whether G 1 is isomorphic to a subgraph of G 2 . Show that the subgraph isomorphism problem is NP-complete.
   證明 NP:檢查 G1 的 adjacency list 中點與邊的關係和 G2 是否一樣。

   證明 NP-hard:此題目為在 G2 中找出完全子圖的問題，所以可以利用解 clique 方法來解，而解 clique 方法為 NP-complete，所以可以得知此問題為 NP-hard。

5. Given an integer m × n matrix A, and an integer m-vector b, the 0-1 integer-programming problem asks whether there exists an integer n-vector x with elements in the set {0, 1} such that Ax ≤ b. Prove that 0-1 integer programming problem is NP- complete.(Hint: Reduce from 3-CNF-SAT.)

   Step1.它是一個 NP 問題
   矩陣 A 和矩陣 x 相乘花費多項式時間。相乘結果和矩陣 b 中各元素大小的比較亦花費多項式時間。

   Step 2.其他屬於 NP 的問題(3-CNF-SAT)可在多項式時間內歸約成它
   轉換方式：[Xn→Xn],[~Xn→(1-Xn)]
   Φ=(X1 v X2 v X3) ^ (X1 v ~X3 v ~X4)
   clause(1):(X1 v X2 v X3) → (X1+X2+X3) ≧1 → (-X1-X2-X3)≦-1
   clause(2):(X1 v ~X3 v ~X4) → (X1+(1-X3)+(1-X4))≧1 → (-X1+X3+X4)≦1
   對應到的矩陣 $A=\begin{bmatrix}-1 & -1 & -1 & 0\\ -1 & 0 & 1 & 1\end{bmatrix}$；$b=\begin{bmatrix}-1\\ 1\end{bmatrix}$
   若存在 x 使滿足 Ax≦b → 3-CNF-SAT 亦有解

6. The longest-simple-cycle problem is the problem of determining a simple cycle (no repeated vertices) of maximum length in a graph. Formulate a related decision problem, and Show that the decision problem is NP-complete.

   要證明 LSC(longest-simple-cycle)是 NP-complete，首先要證明 LSC 為 NP，即在 polynomial time 可以驗證。
   (1) 給定一個 list of vertices 形成一個 LSC，驗證這個 list 的每一個 vertex 沒有重複，可在 polynomial time 完成，再來確定這個 list 上的相鄰兩點在原圖上皆有邊且第一個 vertex 與最後一個 vertex 也有邊，可在 polynomial time 完成。
   所以 LSC∈NP

(2) 再來聯想到 Hamiltonian-cycle 是 LSC 的一個特例，因為當 LSC 的 list 上點數 k=|V|，其實就是在找 Hamiltonian-cycle，也就是 LSC 的 worst case。因此如果一個圖包含 Hamiltonian-cycle，LSC 就是此 Hamiltonian-cycle，因此可將 Hamiltonian-cycle 轉換成 LSC ，HAM-CYCLE<=LSC，且因為 HAM-CYCLE ∈NP，所以 LSC∈NP-complete。

7. Give an efficient greedy algorithm that finds an optimal vertex cover for a tree in linear time.

想法：

(1) 從樹的任意點開始以 DFS 走遍整個樹
(2) 從子節點回來時檢查他和父節點相連的邊有沒有被覆蓋
(3) 如果該條邊沒有被覆蓋就將該條邊的 2 個點都做記號並將父節點加進 Vertex cover 的集合

Pseudocode:

Let G be the tree.
Let visited[ ] be the list that record whether the vertex has been visited.
Let match[ ] be the list that record whether the vertex has been matched.
Let V' be the set of vertex in optimal vertex cover.
Let v be the root of tree G.

```
DFS_modified(G, V)
    visited[v] = true
    for each u ∈ v's children
        if  !visited[u]
            DFS_modified(G, u)
            if  !match[v] && !match[u]
                match[v] = true, match[u] = true
                add v to V'
```