Week 04

名字	到	離
陳大佑		
巫思翰		
梁高銓		
韓欣蓉		

講義整理人：梁高銓

今日 Topic： ch23、ch34、ch35、期末複習(另外一份)

Ch 23 講義：

$$\hat{w}(u,v) = w(u,v) + h(u) - h(v)$$

rewighting

- We have $h(v) \le h(u) + w(u, v)$ if there is no negative weight cycle (why?)
- $\hat{w}(u, v) = w(u, v) + h(u) - h(v) \ge 0$

絕對是正的

1 Computing G', where $G'.V = G.V \cup \{s\}$
and $G'.E = G.E \cup \{(s, v): v \in G.V\}$ and $w(s, v) = 0$.
2 if BELLMAN-FORD(G', w, s)= FALSE
3 print "the input graph contains negative weight cycle"
4 else for each vertex $v \in G'.V$
5 set $h(v)$ to be the value of $\delta(s, v)$ computed by the BF algorithm
6 for each edge $(u, v) \in G'.E$, $\hat{w}(u, v) = w(u, v) + h(u) - h(v)$

```
Let D = (d_uv) be a new n x n matrix
for each vertex u ∈ G.V
    run DIJKSTRA (G, ŵ, u) to compute δ̂(u, v)
        for all v ∈ V[G] .
    for each vertex v ∈ G.V
        d_uv = δ̂(u, v) + h(v) - h(u)
return D
```

Complexity: Using Fibonacci heap $O(V^2 \lg V + VE)$

Using Binary heap implementation: $O(VE \lg V)$

Ch 23 習題：

$\pi_{kj}^{(m-1)}$, if $\min_k \{ W_{kj}^{(m-1)} +$

23.1-7

Suppose that you also want to compute the vertices on shortest paths in the algorithms of this section. Show how to compute the predecessor matrix Π from the completed matrix L of shortest-path weights in $O(n^3)$ time.

簡單來說當前的 shortest path 一定會是 I -> j 或是 經過某個 k 的最短路徑相加，也就是說我們只需要檢查 L[I, j] == L[I, k] + L[k, j]，如果都沒有就是第一種 case，有的話就是我的前一步會更新到 k，預設 PI[I, j] = I。

```less
for i from 1 to n:
    for j from 1 to n:
        if i != j:
            for k from 1 to n:
                if k != i and k != j:
                    # Check if the shortest path i->j passes through k
                    if L[i][j] == L[i][k] + L[k][j]:
                        # Update predecessor of j in the path from i
                        Π[i][j] = Π[k][j]
                        break  # Once a suitable k is found, no need to continue
```

> Modify FASTER-ALL-PAIRS-SHORTEST-PATHS so that it can determine whether the graph contains a negative-weight cycle.

time for each assignment.

```
FASTER-APSP(W, n)
let L and M be new n × n matrices
L = W
r = 1
while r < n − 1
    M = ∞        // initialize M
    EXTEND-SHORTEST-PATHS(L, L, M, n)     // compute M = L^2
    r = 2r
    L = M        // ready for the next iteration
return L
```

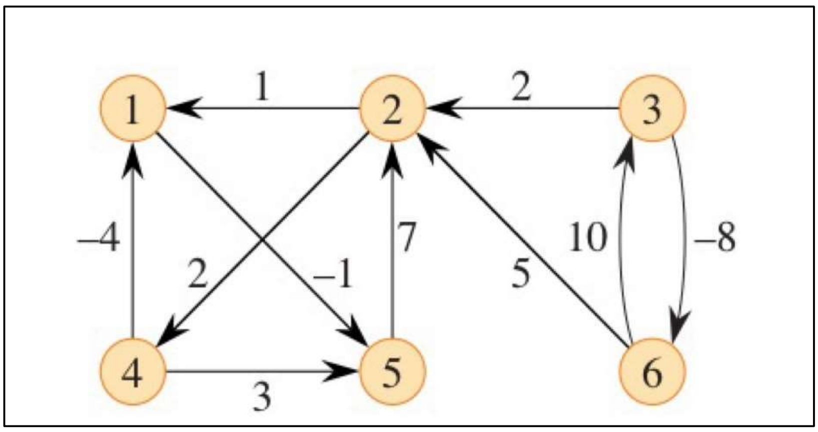
我們知道 2^(k+1) > n-1，因為 2^k > n-1，所以我們可以在多乘一次，也就是在 return 前多加一行，extended-shortest-path(M,M,M',n)，以及 check M' = M or not。

> Give an efficient algorithm to find the length (number of edges) of a minimum-length negative-weight cycle in a graph.

可以利用 APSP 每一次迭代都檢查對角線，時間複雜度是 O(n^4)。

> Run the Floyd-Warshall algorithm on the weighted, directed graph of Figure 25.2. Show the matrix $D^{(k)}$ that results for each iteration of the outer loop.

算算看



建議要

> As it appears above, the Floyd-Warshall algorithm requires $\Theta(n^3)$ space, since we compute $d_{ij}^{(k)}$ for $i, j, k = 1, 2, \ldots, n$. Show that the following procedure, which simply drops all the superscripts, is correct, and thus only $\Theta(n^2)$ space is required.
>
> ```
> FLOYD-WARSHALL'(W)
>     n = W.rows
>     D = W
>     for k = 1 to n
>         for i = 1 to n
>             for j = 1 to n
>                 d[i, j] = min(d[i, j], d[i, k] + d[k, j])
>     return D
> ```

只會根據前一次的來做改變，且改變的時候是 dij^(m+1) = min(dij^m, dik^m + dkj^m)，這代表，我只要在跟心之前，或者是在跟跟心的時候 dik、dkj 都不會被動到就好，那這是對的，因為如果會被動到就代表 dik + dkk < dik 或者是 dkk + dkj < dkj，那這是不可能發生的當圖沒有負環的時候。

> How can we use the output of the Floyd-Warshall algorithm to detect the presence of a negative-weight cycle?

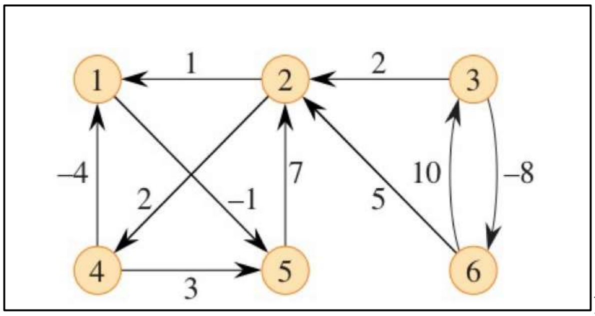
迭代完之後檢查對角線是否有非 0 元素。

23.2-8

Give an $O(VE)$-time algorithm for computing the transitive closure of a directed graph $G = (V, E)$. Assume that $|V| = O(E)$ and that the graph is represented with adjacency lists.

這邊使用 dfs 對於每一個點，但這樣時間複雜度是 O(VE + V*V)，且|V| = O(E)，所以時間複雜就是 O(VE + VE) = O(VE)

> Use Johnson's algorithm to find the shortest paths between all pairs of vertices in the graph of Figure 25.2. Show the values of h and $\hat{w}$ computed by the algorithm.



算算看，建議要

> What is the purpose of adding the new vertex s to V', yielding V'?

This is only important when there are negative-weight cycles in the graph. Using a dummy vertex gets us around the problem of trying to compute $-\infty + \infty$ to find $\hat{w}$. Moreover, if we had instead used a vertex v in the graph instead of the new vertex s, then we run into trouble if a vertex fails to be reachable from v.

> Suppose that $w(u, v) \geq 0$ for all edges $(u, v) \in E$. What is the relationship between the weight functions w and $\hat{w}$?

If all the edge weights are nonnegative, then the values computed as the shortest distances when running Bellman-Ford will be all zero. This is because when constructing G' on the first line of Johnson's algorithm, we place an edge of weight zero from s to every other vertex. Since any path within the graph has no negative edges, its cost cannot be negative, and so, cannot beat the trivial path that goes straight from s to any given vertex. Since we have that $h(u) = h(v)$ for every u and v, the reweighting that occurs only adds and subtracts 0, and so we have that $w(u, v) = \hat{w}(u, v)$

Suppose that we run Johnson's algorithm on a directed graph G with weight function w. Show that if G contains a 0-weight cycle c, then $\hat{w}(u, v) = 0$ for every edge (u, v) in c.

我們知道 rewight 後，一個圈圈的權重不會變，也知道 rewight 後如果沒有負環，他就會都大於等於 0，所以她一定全部都等於 0。

Ch 34 講義：

Cook-Levin Theorem

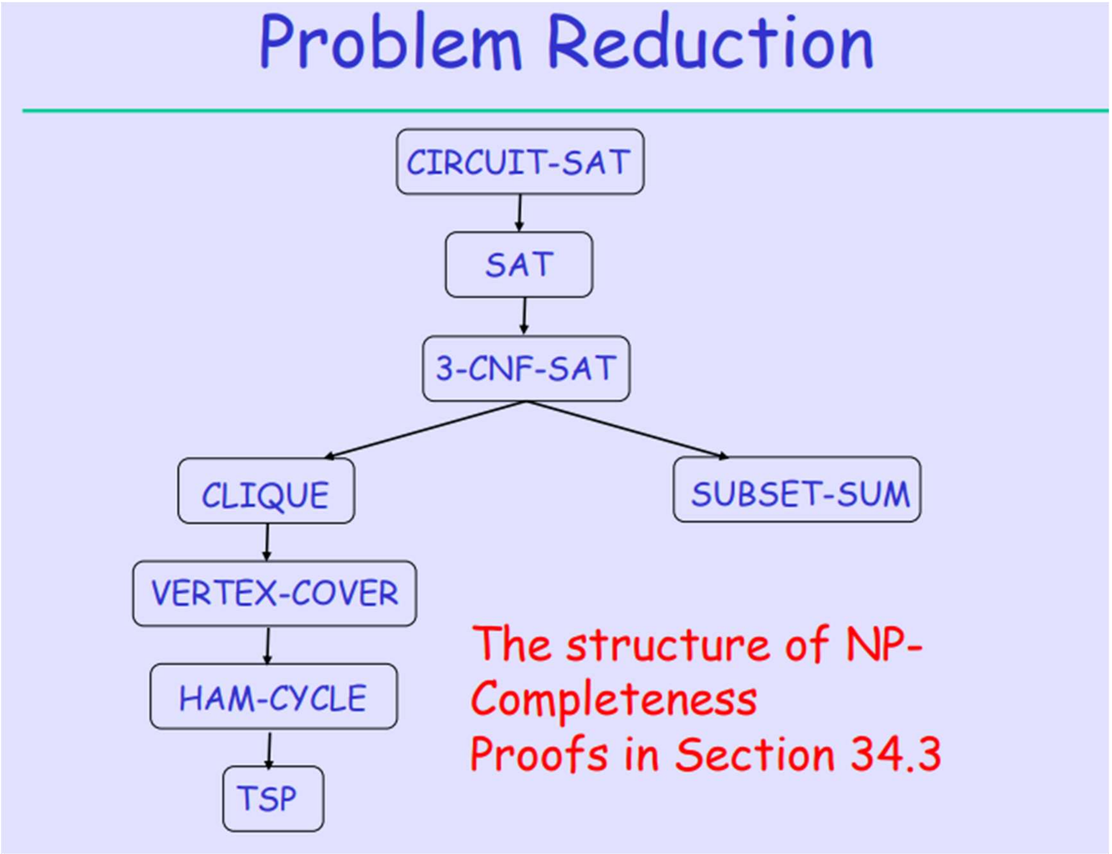
- If SAT is in P, then all problems in NP are also in P

這說明 SAT 是 NP 裡面最難的問題

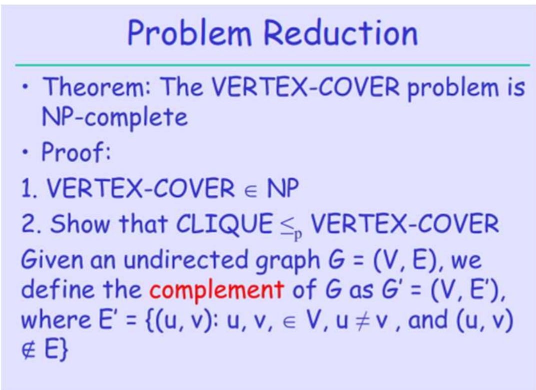
Problem Reduction

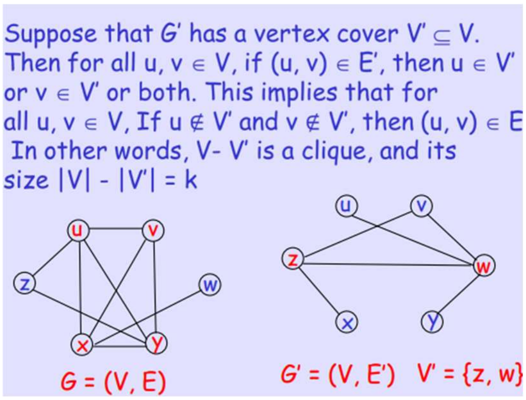
- e.g., A = Finding median, B = Sorting
- We can solve A if we know how to solve B
  - ➔ sorting is as hard as finding median
- eg., A = Topological Sort, B = DFS
- We can solve A if we know how to solve B
  - ➔ DFS is as hard as topological sort

說是 reduction，但是其實是 show 更難



越往下的問題越難，我們可以透過後面的證明，說明如果我解決下面的問題，那上面的問題就被解決了。





我們想要 show 點覆蓋也是 NPC，首先我們先說明他會落在 NP 裡，那它的難度一定小於 NPC，再來要說明另一邊，也就是說，我要說明如果點覆蓋可以解，那某一個 NPC 的問題也可以解，那這邊取的 NPC 問題是點團。

Problem Reduction

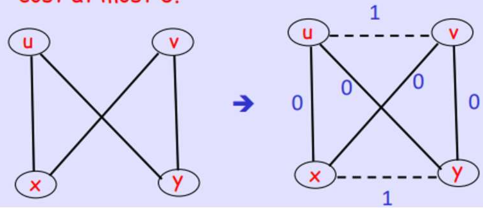
- Theorem: The traveling-salesman problem (TSP) is NP-complete
- Proof:

1. TSP $\in$ NP
2. Show that HAM-CYCLE $\leq_p$ TSP

Let $G = (V, E)$ be an instance of HAM-CYCLE. We construct an instance of TSP as follows. We form a complete graph $G' = (V, E')$, where $E' = \{(i, j) | i, j, \in V, \text{ and } i \neq j\}$ and we define the cost function

$$C(i, j) = \begin{cases} 0 \text{ if } (i,j) \in E \\ 1 \text{ if } (i,j) \notin E \end{cases}$$

We can show that graph G has a Hamiltonian cycle iff graph G' has a tour cost at most 0.



同理，自己想一下。

- NP-Complete: a problem A is in NPC iff (i) A is in NP, and (ii) any problem in NPC can be reduced to it in $O(n^k)$ time.
- If any problem in NPC can be solved in $O(n^k)$ time, then P=NP. It is believed (but not proved) that P $\neq$ NP.
- NP-Hard: a problem A is in NPH iff a problem in NPC can be reduced to A in $O(n^k)$ time.

NPC 的 decision problem，他的最佳化問題是 NPH。

- Give a polynomial algorithm to solve the 2-CNF satisfiability problem

演算法

假設 Boolean formula 有 n 種布林變數，m 個子句

1. 將 Boolean formula 轉成一有向圖 G (Implication Graph)：
   建構方式：
   (1) 對每個 x_i，新增兩個點 X_i 和 $\hat{X}_i$
   (2) 對每個子句 $(x_i \vee x_j)$，新增兩條有向邊 $(\hat{X}_i, X_j)$ 和 $(\hat{X}_j, X_i)$
2. 利用 Kosaraju's Algorithm 找出所有強連通分量。
3. 檢查每組 x_i 和 $\hat{x}_i$ 是否在同一個強連通分量中。如果存在一組相同代表 Unsatisfiable 則回傳 False。若每組 x_i 和 $\hat{x}_i$ 都在不同的強連通分量中，代表此 Boolean formula 為 Satisfiable，回傳 True。

O(V+E) 、 DAG-shotest-path (use topologic sort)

- If a problem is NPC, it cannot be solved by any polynomial time algorithm in the worst cases.
- If a problem is NPC, we have not found any polynomial time algorithm to solve it in the worst cases until now.
- If a problem is NPC, then it is unlikely that a polynomial time algorithm can be found in the future to solve it in the worst cases.

F T F

- If we can prove that the lower bound of an NPC problem is exponential, then we have proved that NP $\neq$ P.
- Any NP-hard problem can be solved in polynomial time if an algorithm can solve the satisfiability problem in polynomial time.

T F

- Suppose that all edge weights in a graph are integers ranging from 1 to |V|. How fast can you make Prim's algorithm run?
- How do you solve the single-source shortest paths problem in directed acyclic graphs (DAGs)?

O(V+E) 、 DAG-shotest-path (use topologic sort)

Ch 34 習題：

> Is the dynamic-programming algorithm for the 0-1 knapsack problem that is asked for in Exercise 16.2-2 a polynomial-time algorithm? Explain your answer.

```
0-1-KNAPSACK(n, W)
    Initialize an (n + 1) by (W + 1) table K
    for i = 1 to n
        K[i, 0] = 0
    for j = 1 to W
        K[0, j] = 0
    for i = 1 to n
        for j = 1 to W
            if j < i.weight
                K[i, j] = K[i - 1, j]
            else
                K[i, j] = max(K[i - 1, j], K[i - 1, j - i.weight] + i.value)
```

這是 fake polynomial，因為如果背包大小大到 2^n，那就是不是 P 了。

> Prove that if G is an undirected bipartite graph with an odd number of vertices, then G is nonhamiltonian.

他會從 A 到 B，但因為是奇數的關係，所以最後停的那一塊一定跟原本的不同塊。

> Show that the hamiltonian-path problem from Exercise 34.2-6 can be solved in polynomial time on directed acyclic graphs. Give an efficient algorithm for the problem.

Use topologic sort，檢查相鄰的是否都有邊連接。

Ch 35 講義：

- Let us consider the following algorithm:
  1. C = an empty set
  2. while (there is an edge in G) {
     3. Pick an edge, say (u,v) ;
     4. Put u and v into C ;
     5. Remove u, v, and all edges adjacent to u or v ;

     }
  6. return C

- What is so special about C?
  - Vertices in C must cover all edges.
  - But … it may not be the smallest one
- How far is it from the optimal?
  - At most two times (why?)
  - Because each edge in line 3 of the algorithm can only be covered by its endpoints ➔ in each iteration, one of the selected vertexes must be in the optimal vertex cover

APPROX_TSP_TOUR(G, c)

1 Select a vertex $r \in G.V$ to be a root vertex

2 grow an MST T for G from root r using MST_PRIM(G, c, r)

3 Let H be the list of vertices visited in a preorder walk of T

4 return the Hamiltonian cycle H that visits the vertices in the order H

Time complexity: $O(E) = O(V^2)$

- Clearly, $|W|=2|T|$. Thus, $|W| \leq 2|H^*|$. Note that W is not a tour. It visits a vertex more than once. However, by triangle inequality, we can delete unnecessary visits to a vertex without increasing the cost to obtain H. In our example, H=(a, b, c, h, d, e, f, g) Thus, $|H| \leq |W| \leq 2|H^*|$. #
- Can we find an approximation algorithm for the general TSP Problem?

3

1. $V_1 = V_2 =$ empty set ;
2. Label the vertices by $x_1, x_2, \ldots, x_n$
3. For ($k = 1$ to n) {

   /* Fix location of x_k */

   Fix x_k to the set such that more in-between edges (with those already fixed vertices $x_1, x_2, \ldots, x_{k-1}$) are obtained ;

   }
4. return the cut (V_1, V_2) ;

- How far is our cut from the optimal?
  - At most two times (why?)
  - When a vertex v is fixed, we will add some edges into the cut and discard some edges (u, v) if u is placed in the same set as v
  - But when each vertex is fixed :

    #edges added $\geq$ #edges discarded

    ➔ total #edges added $\geq m/2$

Ch 35 習題：

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

Algorithm 1 GREEDY-VERTEX-COVER(G)

```
V' = ∅
let L be a list of all the leaves of G
while V ≠ ∅ do
    if |V| == 1 or 0 then
        return V'
    end if
    let v be a leaf of G = (V, E), and {v, u} be its incident edge
    V = V − v
    V' = V' ∪ u
    for each edge {u, w} ∈ E incident to u do
        if d_w == 1 then
            L.insert(w)
        end if
        E = E − {u, w}
    end for
    V = V − u
end while
```

> From the proof of Theorem 34.12, we know that the vertex-cover problem and the NP-complete clique problem are complementary in the sense that an optimal vertex cover is the complement of a maximum-size clique in the complement graph. Does this relationship imply that there is a polynomial-time approximation algorithm with a constant approximation ratio for the clique problem? Justify your answer.

It does not imply the existence of an approximation algorithm for the maximum size clique. Suppose that we had in a graph of size n, the size of the smallest vertex cover was k, and we found one that was size $2k$. This means that in the compliment, we found a clique of size $n-2k$, when the true size of the largest clique was $n-k$. So, to have it be a constant factor approximation, we need that there is some λ so that we always have $n-2k \geq \lambda(n-k)$. However, if we had that k was close to $\frac{n}{2}$ for our graphs, say $\frac{n}{2}-\epsilon$, then we would have to require $2\epsilon \geq \lambda\frac{n}{2}+\epsilon$. Since we can make ϵ stay tiny, even as n grows large, this inequality cannot possibly hold.