Design and Analysis of Algorithms

Final Exam Solution

1. (5%，無限制方法，只有結果對 1 分，只有過程對 4 分)

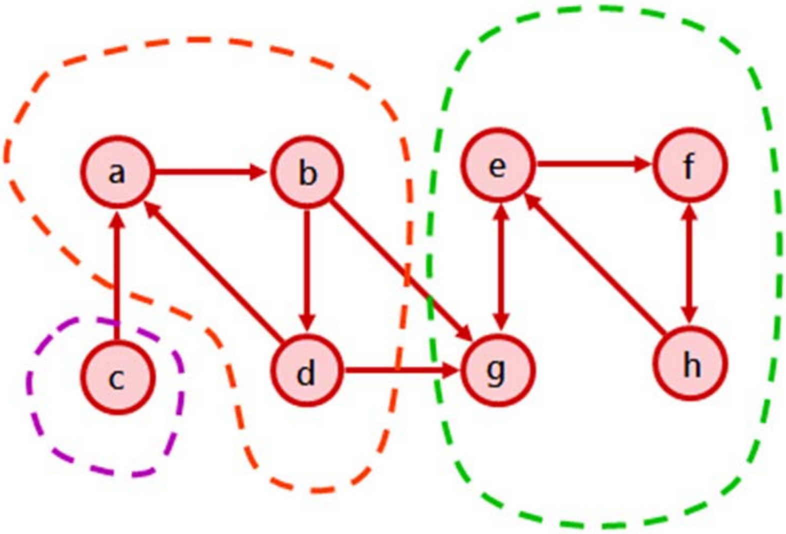
Let m = lg n, thus,T $(2^m) = 3T(2^{m/2}) + m$, (2 分)

We now rename $S(m) = T(2^m)$ to produce new recurrence

$S(m) = 3S(m/2) + m$ (1 分)=> $S(m) = \Theta(m^{\lg 3})$. (1 分)

Changing back to T (n) and resubstituting m = $\lg n$, T (n) = $\Theta((\lg n)^{\lg 3})$ (1 分)

2. (10% , Finding out SCCs : 3 points ; Computational steps : 5 points ; Time complexity : 2 points)



Finding-all-SCC(G)

{

1. Perform DFS on G ;

2. Construct G^T ;

3. while (some node in G^T is undiscovered)

{ u = undiscovered node with latest finishing time refer to Step 1's DFS ;

Perform DFS on G^T from u ;

} // nodes in the DFS tree from u forms an SCC

}

➔ Time-complexity : O(|V|+|E|)

3. (10% , 4 points for each time complexity ; 2 points for the reason.)

Kruskal's algorithm sorts edges in nondecreasing order by weight. If the edge weights are integers in the range 1 to $|V|$, we can use COUNTING-SORT to sort the edges in $\Theta(V + E)$ time (recall COUNTING-SORT correctly sorts n integers in the range 0 to k in $\Theta(n+k)$ time). Then Kruskal's algorithm will run in $O(V+E+V \log V) = O(E+V \log V)$ time.

If the edge weights are integers in the range from 1 to W for some constant W, we can use COUNTING-SORT to sort the edges in $\Theta(W + E)$ time and Kruskal's algorithm will run in $O(W + E + V \log V)$ time.

4. (10% , 過程 4%，結果 3%，time complexity 3%)

Topological sort

1. Call DFS
2. Output the decreasing order of their finishing times.(=結果)

=> Time complexity O(|V|+|E|)

Dijkstra algorithm

1. while (there is unvisited vertex) {
2.   v = unvisited vertex with smallest d ;
3.   Visit v, and Relax all its outgoing edges;
4. }

=> Time complexity O(V^2)

(with binary heap O($E \log V$), with Fibonacci heap O($E + V \log V$))

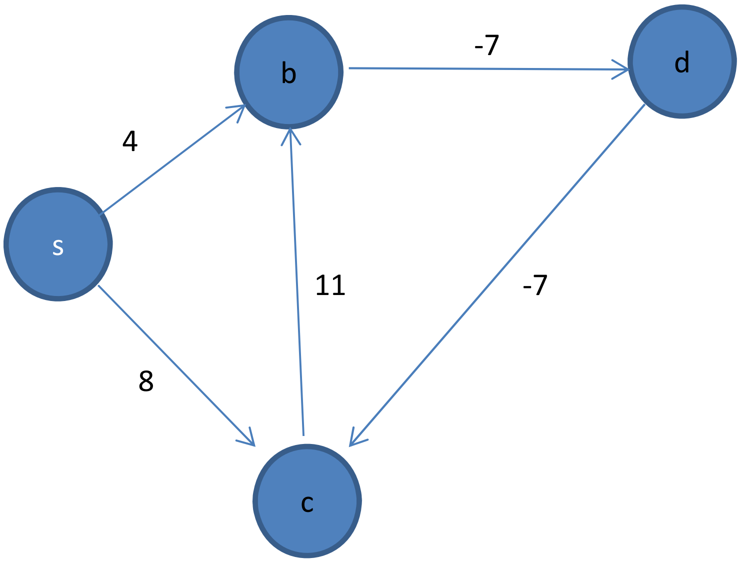
(使用以下方法扣 3 分，因為圖變動就找不到 one-to-all shortest path)

Prim's algorithm, O($E \log E$) = O($E \log V$)

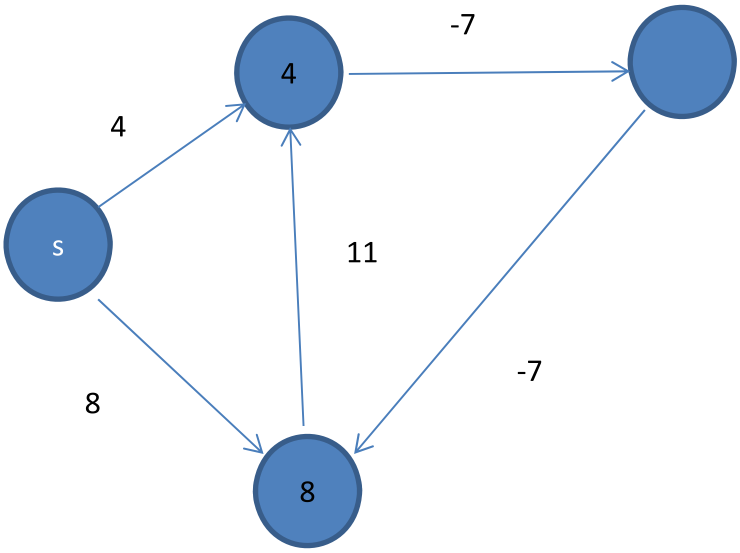
(with binary heap O($E \log V$), with Fibonacci heap O($E + V \log V$))

5. (10% , 部分過程錯誤扣 3 分)

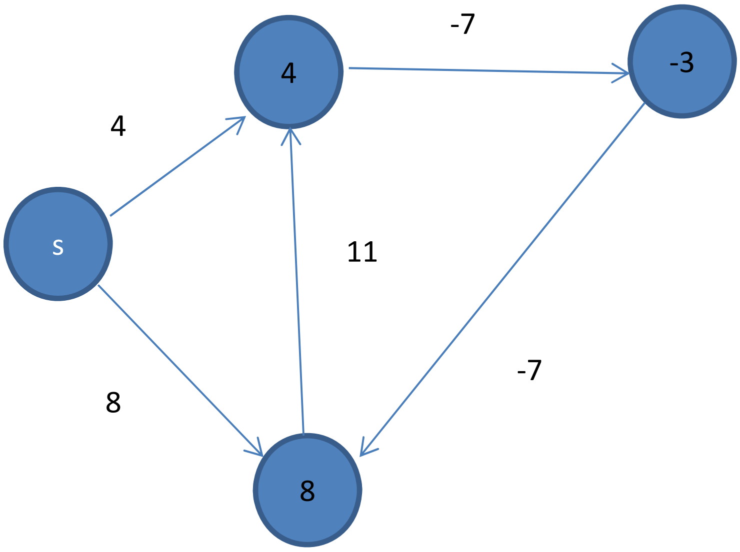
Consider the graph with negative weight cycle like the following example :



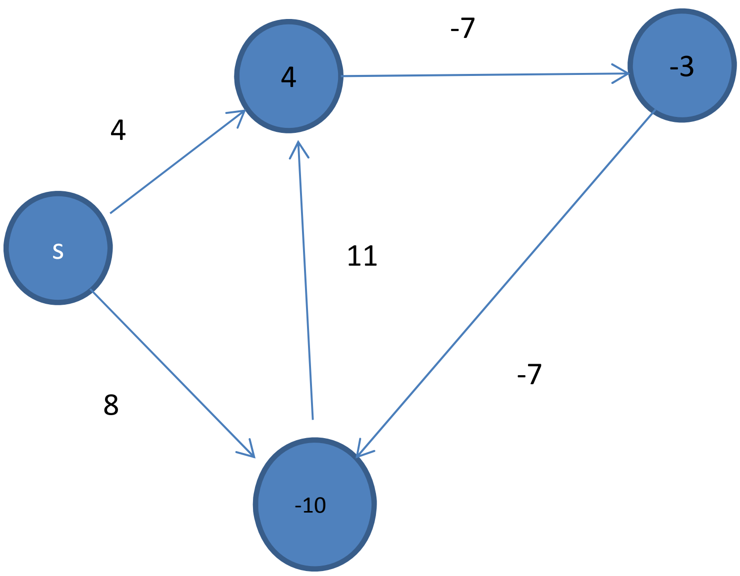
First, we relax vertex b and c :



After including vertex b, we relax vertex d :



After including vertex d, we relax vertex c :



After we including vertex c, we find that vertex b can be relaxed by vertex c.
As a result, if we apply Dijkstra's algorithm to the graph with negative weight cycle , we will not get the correct answer.

6. (10% , algorithm : 4 points ; time complexity : 2 points; matrices : 4 points)

- Algorithm and time complexity: Because there is no negative weight cycle, we choose Floyd-Warshall algorithm

  n = W.rows
  $D^{(0)} = W$
  for k = 1 to n
      Let $D^{(k)} = (d_{ij}^{(k)})$ be a new $n \times n$ matrix
       for i = 1 to n
          for j = 1 to n
              $d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
  return $D^{(n)}$

  Time Complexity: $O(n^3)$

- Apply the method we propose to solve the question

$$D^{(0)} = \begin{matrix} 0 & 3 & 8 & \infty \\ \infty & 0 & \infty & 1 \\ 3 & 4 & 0 & \infty \\ 2 & \infty & -5 & 0 \end{matrix}$$

$$D^{(1)} = \begin{matrix} 0 & 3 & 8 & \infty \\ \infty & 0 & \infty & 1 \\ 3 & 4 & 0 & \infty \\ 2 & 5 & -5 & 0 \end{matrix}$$

$$D^{(2)} = \begin{matrix} 0 & 3 & 8 & 4 \\ \infty & 0 & \infty & 1 \\ 3 & 4 & 0 & 5 \\ 2 & 5 & -5 & 0 \end{matrix}$$

$$D^{(3)} = \begin{matrix} 0 & 3 & 8 & 4 \\ \infty & 0 & \infty & 1 \\ 3 & 4 & 0 & 5 \\ -2 & -1 & -5 & 0 \end{matrix}$$

$$D^{(4)} = \begin{matrix} 0 & 3 & -1 & 4 \\ -1 & 0 & -4 & 1 \\ 3 & 4 & 0 & 5 \\ -2 & -1 & -5 & 0 \end{matrix}$$

7. (10%，部分過程錯誤扣 3 分)

$$\begin{aligned} \hat{w}(p) &= w(p) + h(v_0) - h(v_k) \\ \hat{w}(p) &= \hat{w}(v_{i-1}, v_i) \\ &= (w(v_{i-1}, v_i) + h(v_{i-1}) - h(v_i)) \\ &= w(v_{i-1}, v_i) + h(v_0) - h(v_k) \\ &= w(p) + h(v_0) - h(v_k) \end{aligned}$$

Because $h(v_0)$ and $h(v_k)$ do not depend on the path, if one path from v_0 to v_k is shorter than another using weight function w, then it is also shorter using ŵ. Thus,

$$w(p) = \delta(v_0, v_k) \quad \text{if and only if} \quad \hat{w}(p) = (v_0, v_k)$$

G has a negative-weight cycle using w iff G has a negative-weight cycle using ŵ.
Consider any cycle $C=<v_0,v_1,\ldots,v_k>$ with $v_0=v_k$.
Then $\hat{w}(C) = w(C) + h(v_0) - h(v_k) = w(C)$.

8. (10% , 2 points for each time complexity)

(1) $O(VE \log V)$
(2) $O(VE + V^2 \log V)$
(3) $O(E \log E)$
(4) $O(V^2 \log V + VE)$
(5) $O(V^2E)$
ps. $E = V^2$ 不一定成立，但有做此換算不扣分。

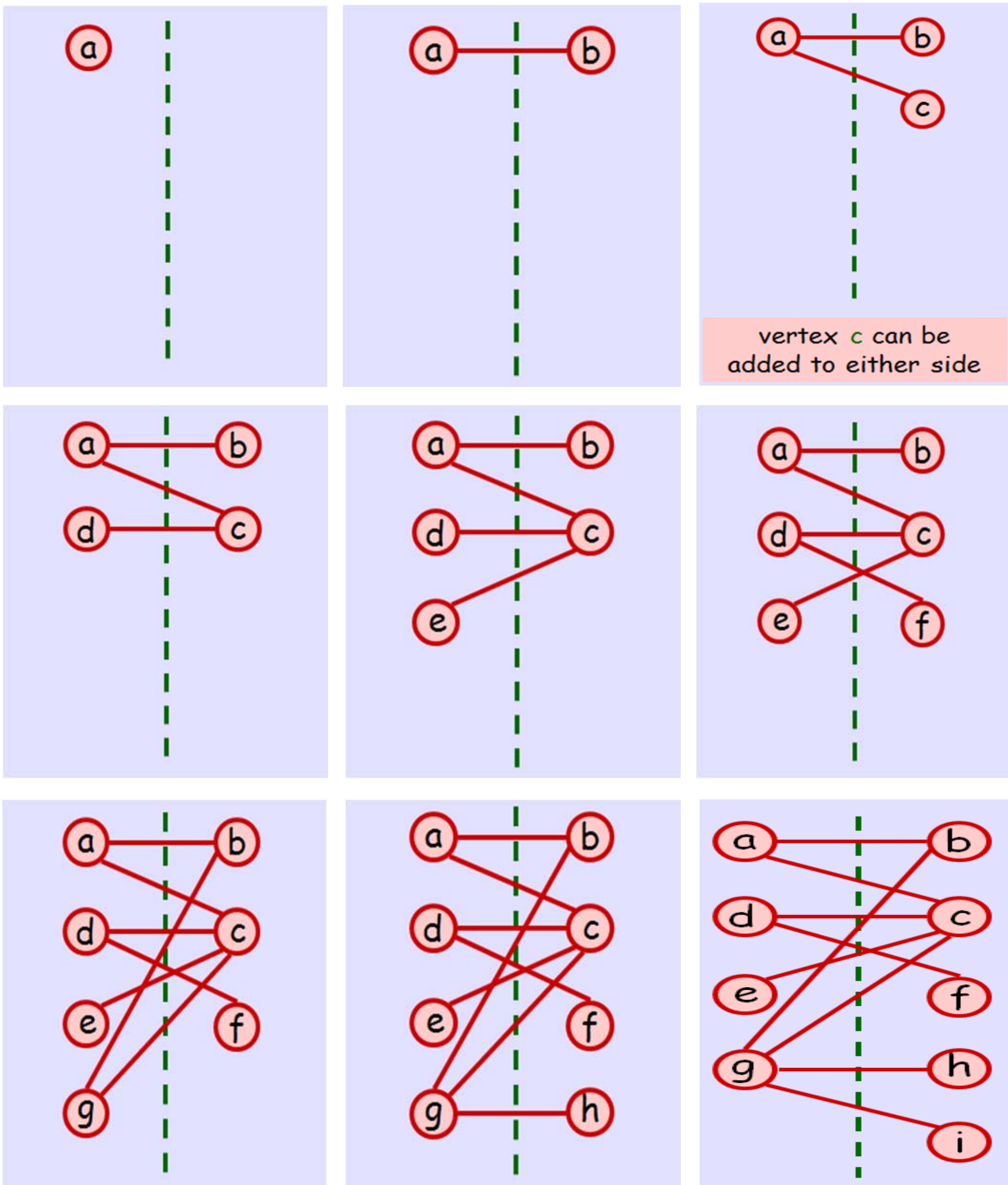
9. (10% , algorithm : 3 points ; steps : 3 points ; proof : 4 points)

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, …, x_{k-1}) are obtained ;

}

4. return the cut (V_1,V_2) ;



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 # of edges added $\geq$ m/2

10. (30%，答案 1 分；解釋 2 分)

a. False, W 可以是 2^K 所以可能跟 n 無關。
b. True, NP-Complete problem can reduce to A in polynomial time, then A is NP-Hard.
c. False, 只知道 SAT 比 A 難，不一定就是 NP-Complete。
d. True, NPC 是 NP 最難的，NPC=P > NP=P
e. True, Halting Problem 比 NP 還難 > NP-Hard
   False, 因為 halting Problem 不在 NP 問題裡。
f. True, 3-CNF SAT 是 NP-Complete, 可以用 P 時間解> NP=P
g. False, 有可能 AB=2 BC=3 CA=4 CA 最短是 4，可是 spanning tree 是 AB+BC=5
h. True, minimum cost edge 一定會在 MST 裡
i. True, 可以 ρ >1 解出來 > P=NP
j. False, NP-hard 不一定全部都在 NP 裡。