Homework 8

林宏陽、周若涓、曾守瑜

Problem1

In the topological sorting algorithm that we discussed in class for directed acyclic graphs, DFS is used to calculate the indegree of each vertex in the input graph. Please give a detailed description of this calculation in adequate pseudocode. You need to define a main routine which invokes the DFS procedure with suitable preWORK and postWORK.

Problem1(cont'd)

```
function MAIN
    for all vertex v do                          ▷ since G is a directed graph
        if v is unmarked then
            DFS(G, v)
function DFS(G, v)
    mark v
    v.indegree := 0                              ▷ preWORK
    for all edge (v, w) do
        if w is unmarked then
            DFS(G, w)
        w.indegree++                             ▷ postWORK
```

Problem2

Given a directed acyclic graph G = (V, E), find a simple (directed) path in G that has the maximum number of edges among all simple paths in G.
The algorithm should run in linear time.

Problem2

```
1   Algorithm FindLongestPath(G)
2   Input: G=(V, E)
3   Output: A stack which records the longest path.
4
5   begin
6       Initialize v.length = 0 & v.Indegree for every vertex v;
7       Max_length=-1
8       for all vertices v in V do
9           if v.Indegree = 0 then
10              put v in Queue;
11              repeat
12                  remove vertex v from Queue;
13                  for all edge(v,w) do
14                      //replace logner path
15                       w.Indegree= w.Indegree - 1
16                      if v.length + 1 > w.length then
17                          w.length := v.length + 1;
18                          w.pre := v;
19                      if w.Indegree=0 then
20                          put w in Queue;
21              until Queue is empty
22      let v of v.length be the largest length in graph G
23          while v.pre is not NULL
24              push v in stack
25              v = v.pre;
26          push v in stack
27  end
```

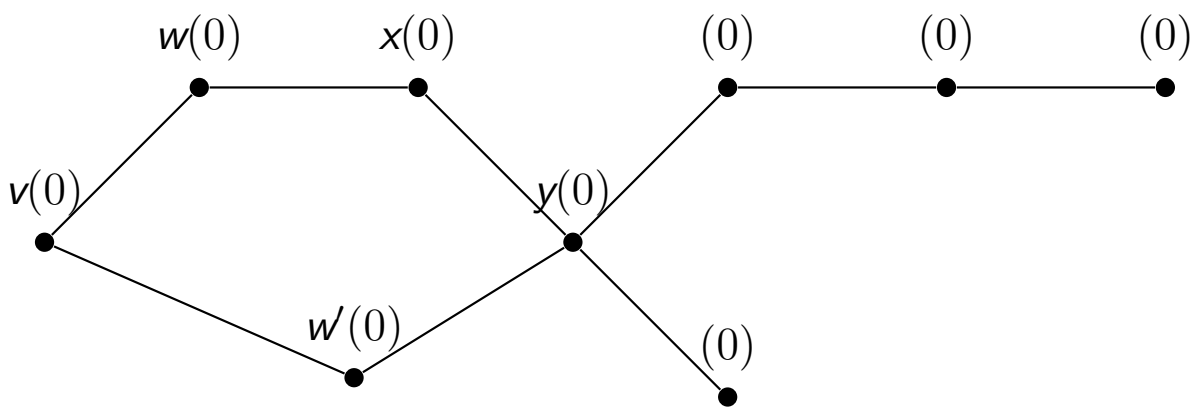
Problem2

```
...
5       begin
6           Initialize v.length = 0 & v.Indegree for every vertex v;
7           Max_length=-1
8           for all vertices v in V do
9               if v.Indegree = 0 then
10                  put v in Queue;
11                  repeat
12                      remove vertex v from Queue;
13                      for all edge(v,w) do
14                          //replace logner path
15                           w.Indegree= w.Indegree - 1
16                          if v.length + 1 > w.length then
17                              w.length := v.length + 1;
18                              w.pre := v;
19                          if w.Indegree=0 then
20                              put w in Queue;
21                  until Queue is empty
22          let v of v.length be the largest length in graph G
23              while v.pre is not NULL
24                  push v in stack
25                  v = v.pre;
26              push v in stack
27      end
```

- 每個 node 的 length 初始化為 0, 並記錄每個 node Indegree value
- 目前的 Max_length value 設為 -1

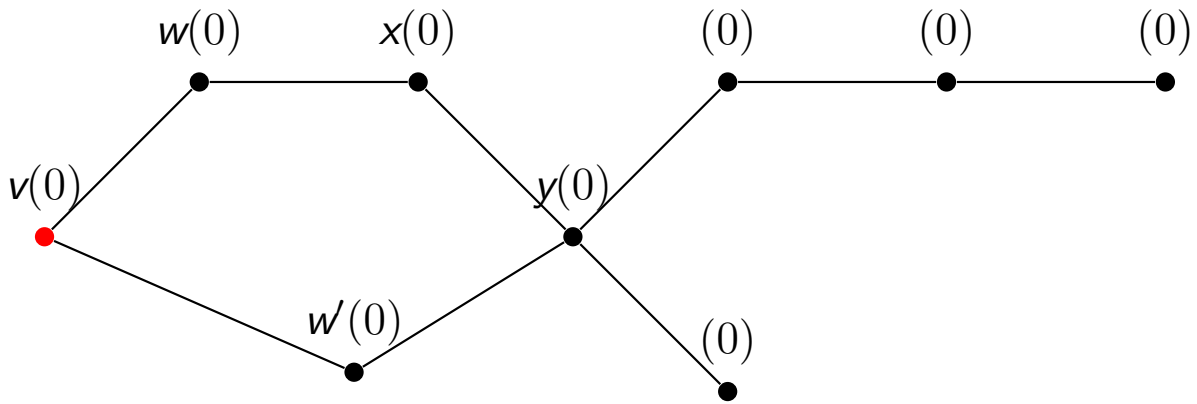
Problem2

```
. . .
5       begin
6           Initialize v.length = 0 & v.Indegree for every vertex v;
7           Max_length=-1
8           for all vertices v in V do
9               if v.Indegree = 0 then
10                  put v in Queue;
11                  repeat
12                      remove vertex v from Queue;
13                      for all edge(v,w) do
14                          //replace logner path
15                           w.Indegree= w.Indegree - 1
16                          if v.length + 1 > w.length then
17                              w.length := v.length + 1;
18                              w.pre := v;
19                          if w.Indegree=0 then
20                              put w in Queue;
21                  until Queue is empty
27      end
```



Problem2

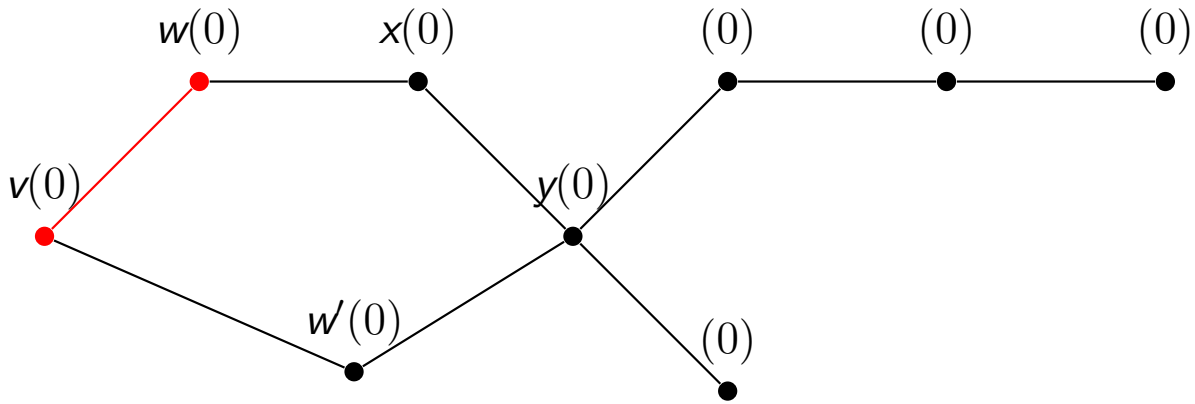
```
. . .
5       begin
. . .
8           for all vertices v in V do
9               if v.Indegree = 0 then /*node v indegree 為 0*/
10                  put v in Queue; /* 將 node v 放入 Queue 中 */
11                  repeat
12                      remove vertex v from Queue;
13                      for all edge(v,w) do
14                          //replace logner path
15                           w.Indegree= w.Indegree - 1
16                          if v.length + 1 > w.length then
17                              w.length := v.length + 1;
18                              w.pre := v;
19                          if w.Indegree=0 then
20                              put w in Queue;
21                  until Queue is empty
. . .
27      end
```



V

Problem2

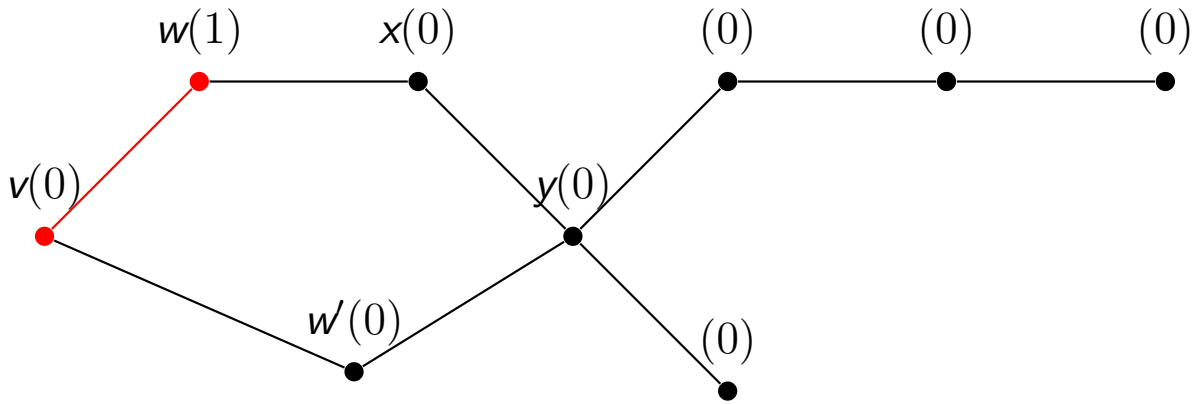
```
...
5       begin
...
8             for all vertices v in V do
9                   if v.Indegree = 0 then
10                        put v in Queue;
11                        repeat
12                              remove vertex v from Queue; /* 將 node v 從 Queue 中移出 */
13                              for all edge(v,w) do /*edge(v,w)、edge(v,w'), 先處理 edge(v,w)*/
14                                    //replace logner path
15                                     w.Indegree= w.Indegree - 1 /*w.Indegree=1-1=0*/
16                                    if v.length + 1 > w.length then /*v.length + 1 > w.length is 0+1>1*/
17                                          w.length := v.length + 1;
18                                          w.pre := v;
19                                    if w.Indegree=0 then
20                                          put w in Queue;
21                        until Queue is empty
...
27      end
```



$\emptyset$

Problem2

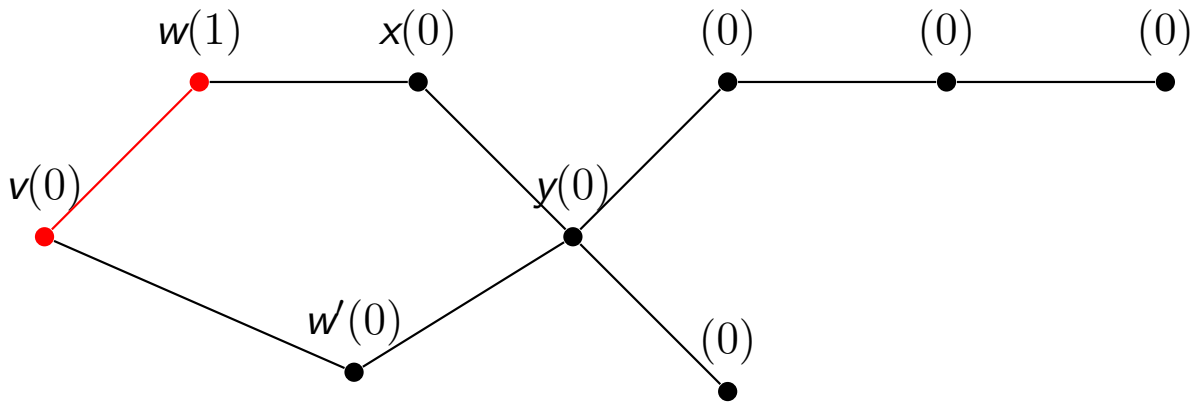
```
. . .
5       begin
. . .
8             for all vertices v in V do
9                   if v.Indegree = 0 then
10                        put v in Queue;
11                        repeat
12                              remove vertex v from Queue;
13                              for all edge(v,w) do
14                                    //replace logner path
15                                     w.Indegree= w.Indegree - 1
16                                    if v.length + 1 > w.length then
17                                          w.length := v.length + 1;/* w.length = v.length + 1=0+1=1*/
18                                          w.pre := v; /* w 前一個 node 設為 v*/
19                                    if w.Indegree=0 then /* w 的 Indegree 為 0*/
20                                          put w in Queue;
21                        until Queue is empty
. . .
27      end
```



$\emptyset$

Problem2

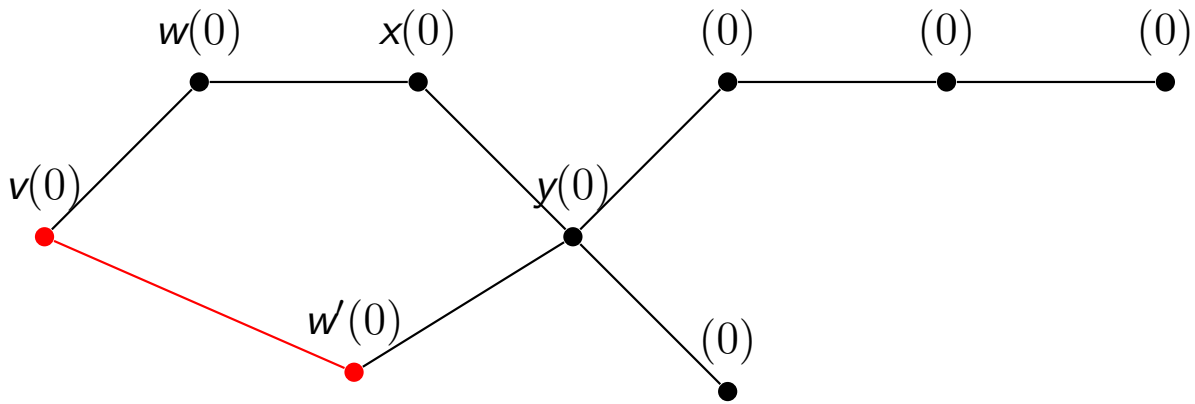
```
. . .
5       begin
. . .
8           for all vertices v in V do
9               if v.Indegree = 0 then
10                  put v in Queue;
11                  repeat
12                      remove vertex v from Queue;
13                      for all edge(v,w) do
14                          //replace logner path
15                           w.Indegree= w.Indegree - 1
16                          if v.length + 1 > w.length then
17                              w.length := v.length + 1;
18                              w.pre := v;
19                          if w.Indegree=0 then
20                              put w in Queue; /* 將 w 放入 Queue*/
21                  until Queue is empty
. . .
27      end
```



W

Problem2

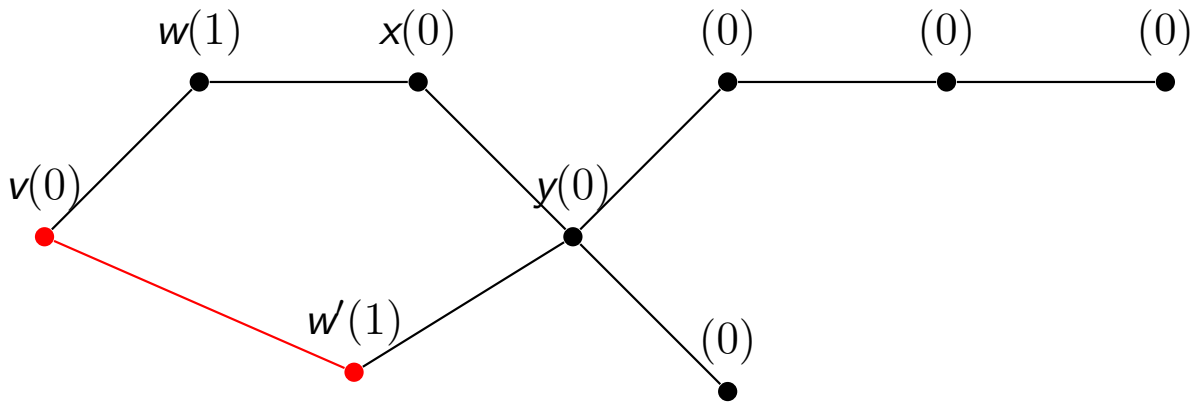
```
. . .
5       begin
. . .
8           for all vertices v in V do
9               if v.Indegree = 0 then
10                  put v in Queue;
11                  repeat
12                      remove vertex v from Queue;
13                      for all edge(v,w) do /* 再來處理 edge(v,w')*/
14                          //replace logner path
15                           w.Indegree= w.Indegree - 1 /*w'.Indegree=1-1=0*/
16                          if v.length + 1 > w.length then /*v.length + 1 > w'.length is 0+1>1*/
17                              w.length := v.length + 1;
18                              w.pre := v;
19                          if w.Indegree=0 then
20                              put w in Queue;
21                  until Queue is empty
. . .
27      end
```



W

Problem2

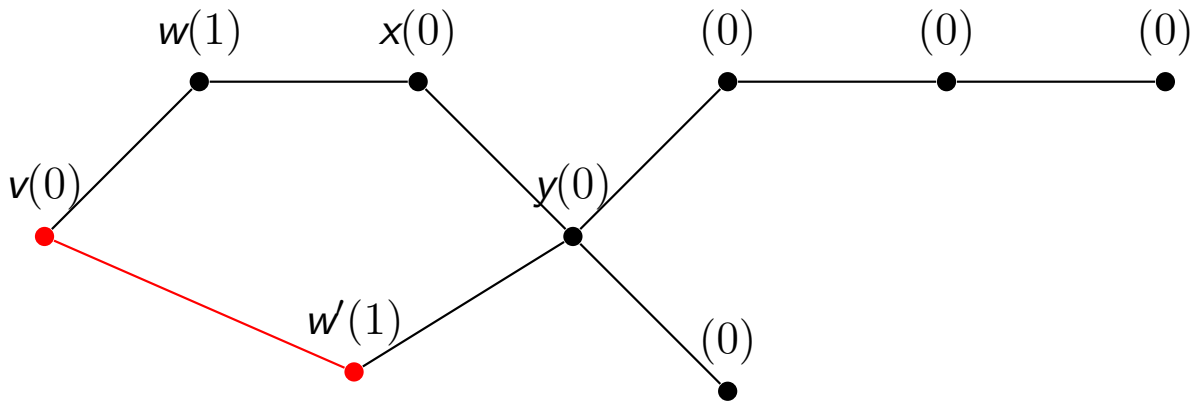
```
. . .
5       begin
. . .
8             for all vertices v in V do
9                   if v.Indegree = 0 then
10                        put v in Queue;
11                        repeat
12                              remove vertex v from Queue;
13                              for all edge(v,w) do
14                                    //replace logner path
15                                     w.Indegree= w.Indegree - 1
16                                    if v.length + 1 > w.length then
17                                          w.length := v.length + 1;/* w'.length = v.length + 1=0+1=1*/
18                                          w.pre := v; /* w' 前一個 node 設為 v*/
19                                    if w.Indegree=0 then /* w' 的 Indegree 為 0*/
20                                          put w in Queue;
21                        until Queue is empty
. . .
27      end
```



W

Problem2

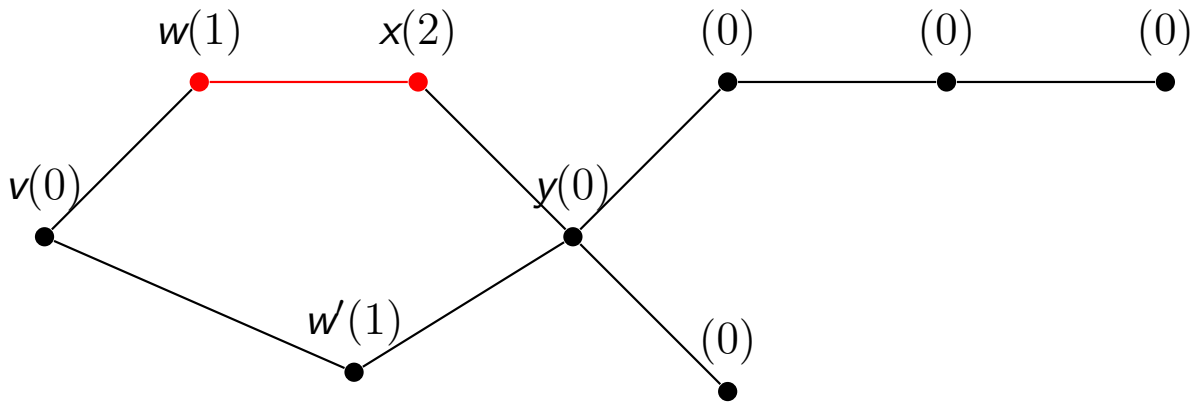
```
...
5       begin
...
8           for all vertices v in V do
9               if v.Indegree = 0 then
10                  put v in Queue;
11                  repeat
12                      remove vertex v from Queue;
13                      for all edge(v,w) do
14                          //replace logner path
15                           w.Indegree= w.Indegree - 1
16                          if v.length + 1 > w.length then
17                              w.length := v.length + 1;
18                              w.pre := v;
19                          if w.Indegree=0 then
20                              put w in Queue; /* 將 w' 放入 Queue*/
21                  until Queue is empty
...
27      end
```



w, w'

Problem2

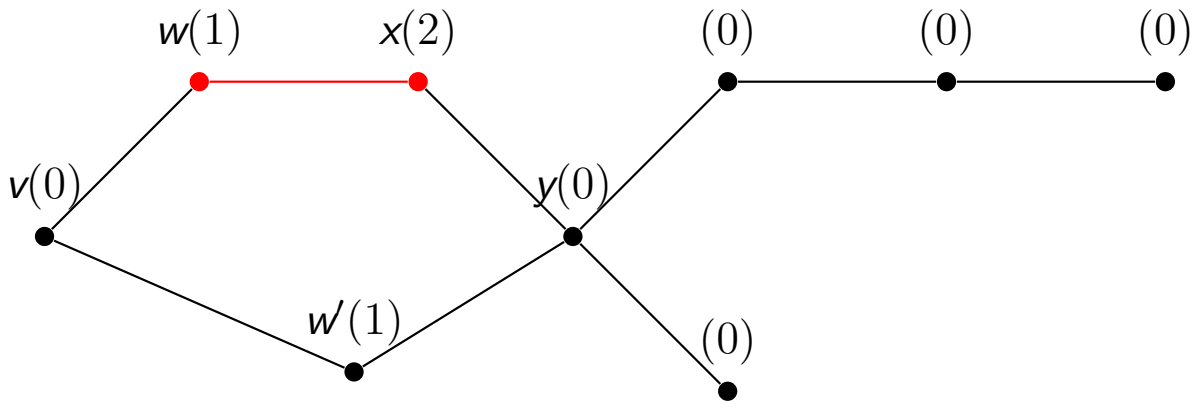
```
. . .
5       begin
. . .
8           for all vertices v in V do
9               if v.Indegree = 0 then
10                  put v in Queue;
11                  repeat
12                      remove vertex v from Queue; /* 將 w 移出 Queue*/
13                      for all edge(v,w) do /* 處理 edge(w,x)*/
14                          //replace logner path
15                           w.Indegree= w.Indegree - 1 /*x 的 indegree 變成 0*/
16                          if v.length + 1 > w.length then
17                              w.length := v.length + 1; /*x 的 length 為 w.length+1=2 */
18                              w.pre := v; /*x 的前一個 node 設成 w*/
19                          if w.Indegree=0 then
20                              put w in Queue;
21                  until Queue is empty
. . .
27      end
```



w'

Problem2

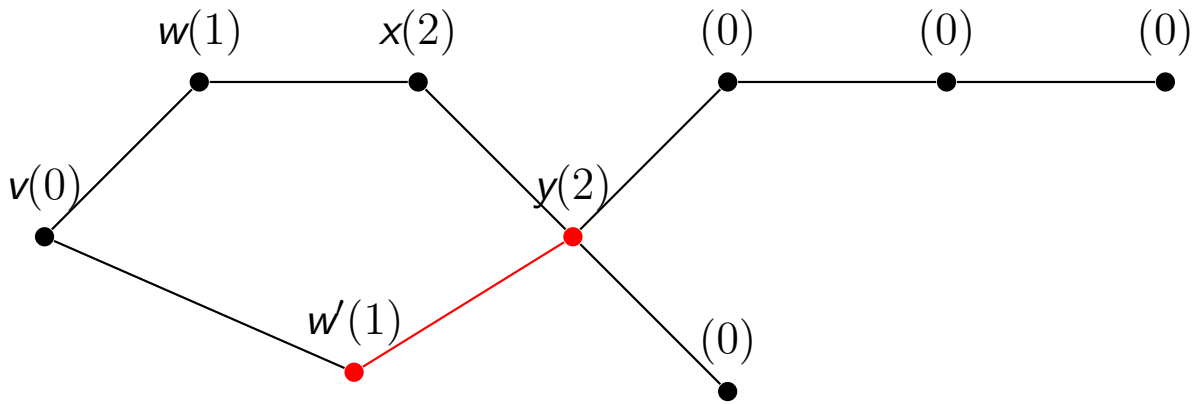
```
...
5       begin
...
8           for all vertices v in V do
9               if v.Indegree = 0 then
10                  put v in Queue;
11                  repeat
12                      remove vertex v from Queue;
13                      for all edge(v,w) do
14                          //replace logner path
15                           w.Indegree= w.Indegree - 1
16                          if v.length + 1 > w.length then
17                              w.length := v.length + 1;
18                              w.pre := v;
19                          if w.Indegree=0 then /*x 的 indegree 為 0*/
20                              put w in Queue; /* 將 x 放入 Queue*/
21                  until Queue is empty
...
27      end
```



w', x

Problem2

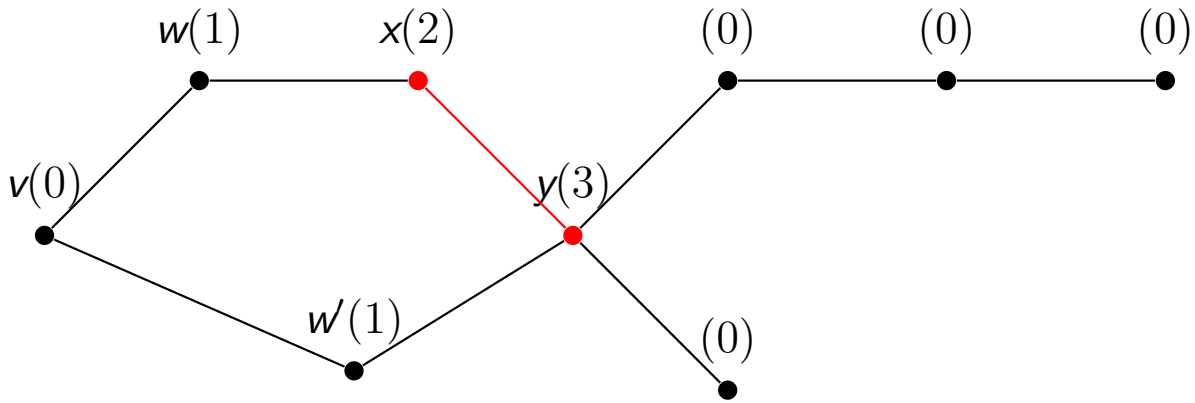
```
. . .
5       begin
. . .
8           for all vertices v in V do
9               if v.Indegree = 0 then
10                  put v in Queue;
11                  repeat
12                      remove vertex v from Queue; /* 將 w' 移出 Queue*/
13                      for all edge(v,w) do /* 處理 edge(w',y)*/
14                          //replace logner path
15                           w.Indegree= w.Indegree - 1 /*y 的 indegree 變成 1*/
16                          if v.length + 1 > w.length then
17                              w.length := v.length + 1; /*y 的 length 為 w'.length+1=2 */
18                              w.pre := v; /*y 的前一個 node 設成 w'*/
19                          if w.Indegree=0 then
20                              put w in Queue;
21                  until Queue is empty
. . .
27      end
```



X

Problem2

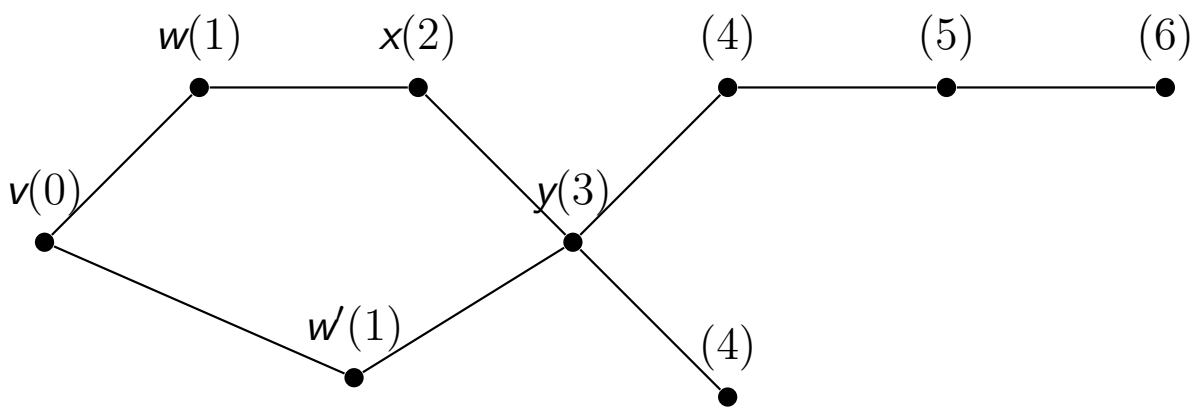
```
. . .
5       begin
. . .
8           for all vertices v in V do
9               if v.Indegree = 0 then
10                  put v in Queue;
11                  repeat
12                      remove vertex v from Queue; /* 將 x 移出 Queue*/
13                      for all edge(v,w) do /* 處理 edge(x,y)*/
14                          //replace logner path
15                           w.Indegree= w.Indegree - 1 /*y 的 indegree 變成 0*/
16                          if v.length + 1 > w.length then
17                              w.length := v.length + 1; /*y 的 length 為 x.length+1=3 */
18                              w.pre := v; /*y 的前一個 node 設成 x*/
19                          if w.Indegree=0 then /*y indegree 為 0*/
20                              put w in Queue; /*y push 入 stack*/
21                  until Queue is empty
. . .
27      end
```



y

Problem2

```
...
5       begin
...
8           for all vertices v in V do
9               if v.Indegree = 0 then
10                  put v in Queue;
11                  repeat
12                      remove vertex v from Queue;
13                      for all edge(v,w) do
14                          //replace logner path
15                           w.Indegree= w.Indegree - 1
16                          if v.length + 1 > w.length then
17                              w.length := v.length + 1;
18                              w.pre := v;
19                          if w.Indegree=0 then
20                              put w in Queue;
21                  until Queue is empty
...
27      end
```



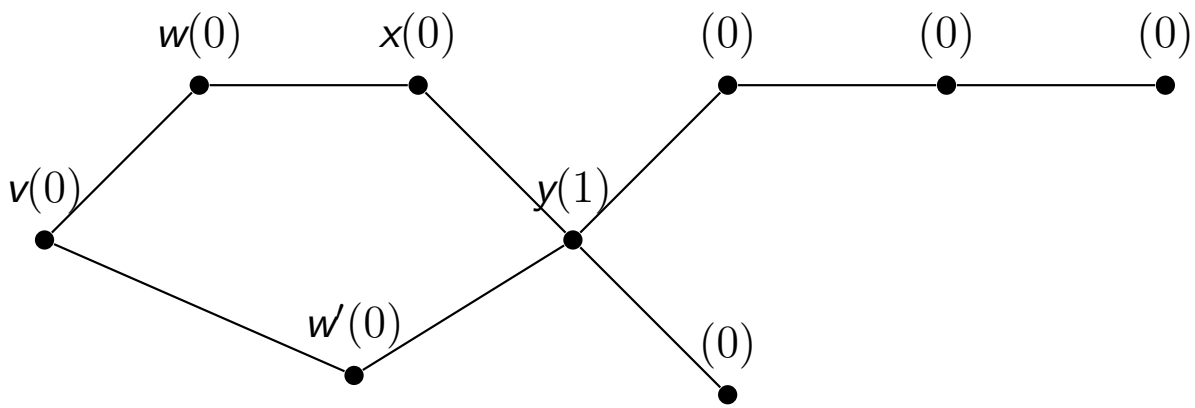
Problem2

```
...
5       begin
...
8             for all vertices v in V do
9                   if v.Indegree = 0 then
10                        put v in Queue;
11                        repeat
12                              remove vertex v from Queue;
13                              for all edge(v,w) do
14                                    //replace logner path
15                                     w.Indegree= w.Indegree - 1
16                                    if v.length + 1 > w.length then
17                                          w.length := v.length + 1;
18                                          w.pre := v;
19                                    if w.Indegree=0 then
20                                          put w in Queue;
21                        until Queue is empty
22              let v of v.length be the largest length in graph G
23                    while v.pre is not NULL
24                        push v in stack
25                        v = v.pre;
26                    push v in stack
27      end
```

- v 為 G 中最長的 length value
- 只要 v 的 pre 還有 node 就將 v push 入 stack
- v 移到前一個 node
- 只要 v 的 pre 沒有 node 就跳出 while loop
- 並將 v push 入 stack

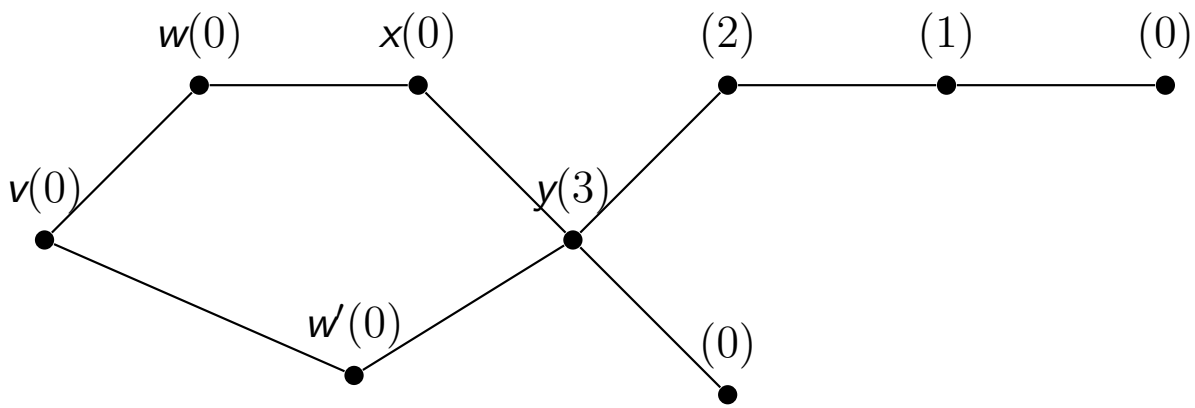
Problem2: 法 2(DFS)

```
1       Algorithm FindLongestPath(G)
2       Input: G=(V, E)
3       Output: A stack which records the longest path.
4
5       preWORK: v.length=0
6       postWORK:
7             if w.length + 1 > v.length /*traverse from right node to left node*/
8                   v.length = w.length +1
9                   v.pos = w;
10
11       let v of v.length be the largest length in graph G
12             while v.pos is not NULL
13                   push v in stack
14                   v = v.pos;
15             push v in stack
```



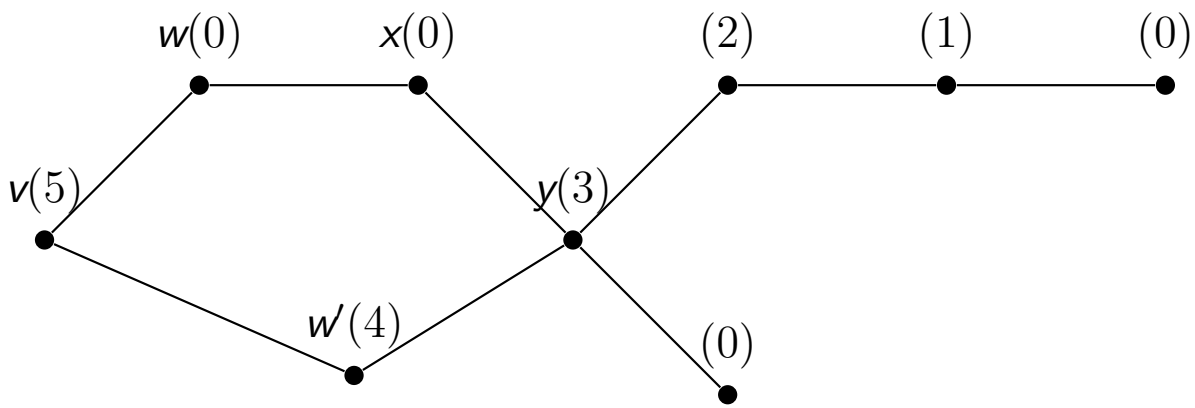
Problem2: 法 2(DFS)

```
1       Algorithm FindLongestPath(G)
2       Input: G=(V, E)
3       Output: A stack which records the longest path.
4
5       preWORK: v.length=0
6       postWORK:
7             if w.length + 1 > v.length /*traverse from right node to left node*/
8                   v.length = w.length +1
9                   v.pos = w;
10
11       let v of v.length be the largest length in graph G
12             while v.pos is not NULL
13                   push v in stack
14                   v = v.pos;
15             push v in stack
```



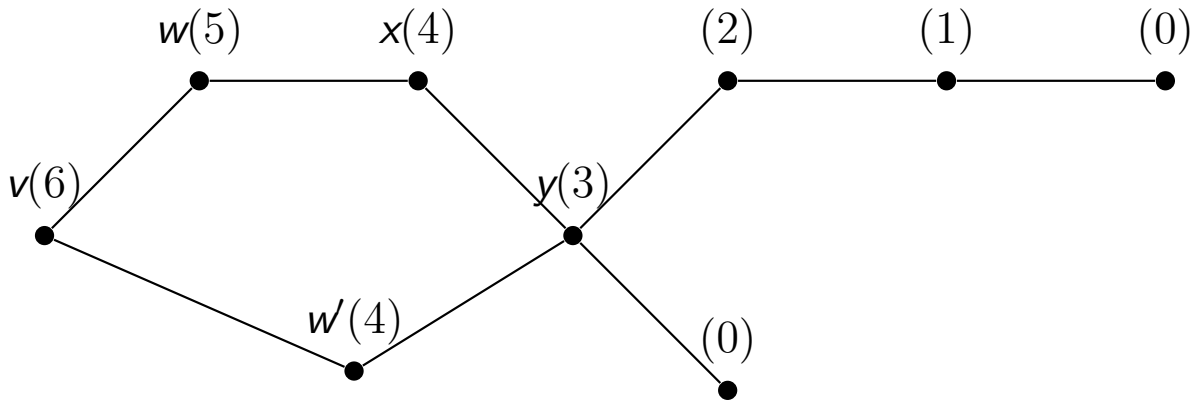
Problem2: 法 2(DFS)

```
1       Algorithm FindLongestPath(G)
2       Input: G=(V, E)
3       Output: A stack which records the longest path.
4
5       preWORK: v.length=0
6       postWORK:
7             if w.length + 1 > v.length /*traverse from right node to left node*/
8                   v.length = w.length +1
9                   v.pos = w;
10
11       let v of v.length be the largest length in graph G
12             while v.pos is not NULL
13                   push v in stack
14                   v = v.pos;
15             push v in stack
```



Problem2: 法 2(DFS)

```
. . .
5       preWORK: v.length=0
6       postWORK:
7             if w.length + 1 > v.length /*traverse from right node to left node*/
8                   v.length = w.length +1
9                   v.pos = w;
10
11       let v of v.length be the largest length in graph G
12            while v.pos is not NULL
13                  push v in stack
14                  v = v.pos;
15            push v in stack
```



- v 為 G 中最長的 length value
- 只要 v 的 pos 還有 node 就將 v push 入 stack
- v 移到下一個 node
- 只要 v 的 pos 沒有 node 就跳出 while loop
- 並將 v push 入 stack
- 優點: 不需額外的 queue

Problem3

Consider Dijkstra's algorithm for single-source shortest paths. The values of SP for all vertices may be stored in either an array or a heap. How do these two implementations compare in terms of time complexity? Please explain.

Problem3(cont'd)

1. Using array:
   For each vertex v, we need to find unvisited adjacent vertex w and the weight of edge (v, w) is minimal. This takes $O(|V|^2)$ time.
   In addition, we need to update the value of SP after picking the cloest vertex, this takes $O(|E|)$ time.
   The total run time using array is $O(|E| + |V|^2) = O(|V|^2)$.
   (because $|E|$ is $O(|V|^2)$ in a simple graph)

Problem3(cont'd)

2. Using heap(Assume it is a binary min-heap):
   Building the heap takes $O(|V|)$ time. For each vertex v, in order to find the closest unvisited adjacent vertex, the only thing needs to do is deleting the heap root and rebuild(heapify) it because the heap root stands for the minimal SP from the source. This takes $O(|V| \log |V|)$ time.
   In addtion, we need to update the value of SP after picking the cloest vertex, which takes $O(|E|)$ time. Beacuse this happens in a heap, it takes $O(|E| \log |V|)$ time totally.
   The total run time using binary heap is $O((|E| + |V|) \log |V|)$.

   With the help of Fibonacci heap, it improves this to $O(|E| + |V| \log |V|)$ (see wikipedia).

Problem4

Prove that if the costs of all edges in a given connected graph are distinct, then the graph has exactly one unique minimum-cost spanning tree.

Problem4

- 利用反證法
- **若存在有兩顆相異的 minimum-cost spanning trees MST_1(G) 與 MST_2(G)**
- MST_1(G) 與 MST_2(G) 的 edges 按 cost 大小排序
  - MST_1(G): e1, e2,$\cdots$, MST_2(G): e1$'$, e2$'$,$\cdots$
- 假設 e_i be the minimum cost edge in MST_1 but not MST_2(G).
- 假設 e_i' be the minimum cost edge in MST_2(G) but not MST_1 .
- 假設 $e_i < e_i$', then MST_2(G)$\cup \{e_i\}$ 會產生 cycle
- Let e_k' (that is in MST_2(G)) be the maximum cost edge of the cycle

Problem4

- 因為每個 edge 的 cost 都是相異的且 e_k' 是 maximum cost edge
- e_k' 不屬於任何 minimum-cost spanning trees ，但 e_k' 出現在 MST_2(G) 中
- MST_2(G) 不是 minimum-cost spanning tree
- 這違反假設 **若存在有兩顆相異的 minimum-cost spanning trees MST_1(G) 與 MST_2(G)**
- 故不可能有兩棵 minimum-cost spanning trees

Problem5

給定圖 G 以及 minimum-cost spanning tree T
若是 G 的某個 edge 的 cost 產生變化
如何找出新的 minimum-cost spanning tree ?

別忘了分析複雜度哦

Problem5

首先很明顯地：

	變大	變小
在 T 中	要檢查	不用檢查
不在 T 中	不用檢查	要檢查

我們還需要 Lemma：
將 spanning tree 上沒有連線的兩點連在一起，則會形成一個 cycle
而對 minimum-cost spanning tree 而言，加上另外一個 edge 形成 cycle 時，這個 edge 一定是 cycle 當中 cost 最大的 (若否，則去除另外一個 edge 才能得到真正的 minimum-cost spanning tree)

Problem5

若 tree 上的 edge cost 升高
假設我們去除了這個 edge，就會將樹分成兩塊
找出所有連接著兩塊樹的 edges，選出其中 cost 最小的

Problem5

Is this a minimum-cost spanning tree?
選定一個不在新樹中的 edge
若它與其他 edge 形成的 cycle 不包含新的 edge
那麼可以用原本的 minimum-cost spanning tree 套入 Lemma 得知
這個 edge 不好；
若 cycle 包含新 edge
代表這個 edge 也是連接兩塊小樹的 edge
而這個 edge 之所以當初沒被選中就是因為不夠好

所以現在的這個樹確實已經是最好的了

Problem5

若 tree 外的 edge cost 降低
假設我們把這個 edge 加到 T，就會產生 cycle
找出 cycle 當中 cost 最高的去掉

Problem5

Is this a minimum-cost spanning tree?
選定一個不在新樹中的 edge
觀察原本的 tree 與之形成的 cycle
這個 edge 的 cost 必然會大於被取代的 edge
而新 edge 本身 cost 就小於被取代的 edge
所以在新樹中這條 edge 形成的 cycle 中
也必然是這條 edge cost 最高

所以現在的這個樹確實已經是最好的了

Problem5

```
function FindNewMinCostSpanningTree(G, T, e = (v, w))
    if e.cost increase and e in T then
        remove e from T
        do DFS on T from v and mark 1
        do DFS on T from w and mark 2
        for all (x, y) and x.mark ≠ y.mark do
            if (x, y) is smaller then
                newEdge := (x, y);
        add newEdge into T
    else if e.cost decrease and e not in T then
        add e into T
        find cycle and store the cycle in C
        for all (x, y) in C do
            if (x, y) is larger then
                removeEdge := (x, y);
        remove removeEdge from T
```

Problem5

複雜度分析：
兩個子樹的 DFS 總共花費 O(|V|+|E|)
遍歷所有 edge 花費 O(|E|)

找尋 cycle 花費 O(|V|+|E|)
遍歷 cycle 上的 edge 則花費 O(|V|) （ 不是 O(|E|) ）