Q1

找出頂點 s 到 t 的路徑數

先將整個圖做拓撲排序，找出 s (V0)和 t (Vn)之間的所有頂點 (V1,V2,V3,…)，並將t頂點的路徑數設為 1 ( 自己到自己只有一條 )，從 t 頂點往 s 走。

對於每個頂點 V 的路徑數 = 相鄰頂點的路徑數和。

最後回到 V0 時，即是 s 到 t 的所有路徑數。

Pseudocode

SIMPLE-PATHS(G, u, v)  
 TOPO-SORT(G)  
 let {v[1], v[2]..v[k - 1]} be the vertex between u and v  
 v[0] = u & v[k] = v  
 for j = 0 to k – 1  
 DP[j] = ∞  
 DP[k] = 1  
 return SIMPLE-PATHS-AID(G, DP, 0)

SIMPLE-PATHS-AID(G, DP, i)  
 if i > k  
 return 0  
 else if DP[i] != ∞  
 return DP[i]  
 else  
 DP[i] = 0  
 for v[m] in G.adj[v[i]] and 0 < m ≤ k  
 DP[i] += SIMPLE-PATHS-AID(G, DP, m)  
 return DP[i]

Time Complexity

Topological Sorting -> O( V + E ) & SIMPLE-PATHS-AID -> O( E ) = O ( V + E )

Q2

**DFS(G)**

Initailly set c[u]<-White , p[u]<- NUll for each u∈V

for each u∈V do if c[u] = White then Visit(u)  
Visit(u){  
　　c[u]<-Black  
　　for each ( v∈Adj[u] && v!=p[u] ){  
　　　if (c[v]==Black)　there is a cycle in G  
　　　else{   
　　　　p[v]<-u  
　　　　Visit(v)  
　　　}  
　　}  
}

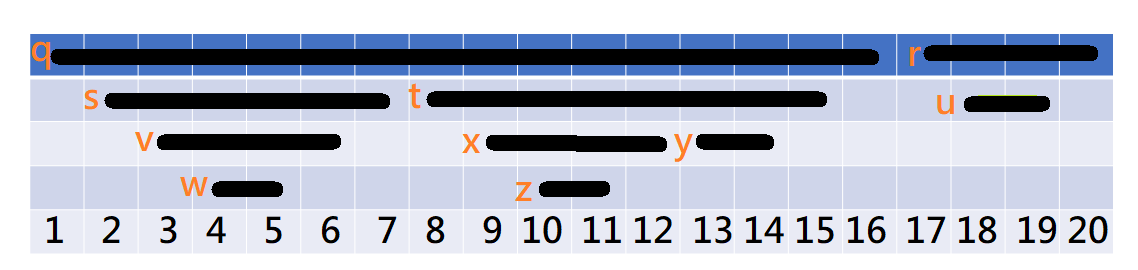
Time complexity: O( lVl )

Q3

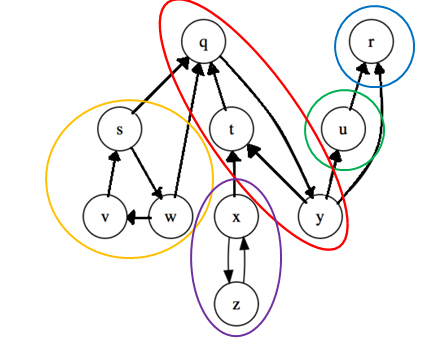
visit(u)  
　　c[u] = gray  
　　d[u] = ++time  
　　for each v ∈ adj[u]   
　　　　if c[v] = white : edge[u,v] = tree\_edge  
　　　　if c[v[ = gray : edge [u,v] = back\_edge  
　　　　else  
　　　　　if d[v]<d[u] : edge [u,v] = cross\_edge   
　　　　　// (u, v) is a cross edge) ⇔ d[v] < f[v] < d[u] < f[u]  
 　　else : edge[u,v] = forward\_edge  
　　　　if c[v] = white   
　　　　　　p[v] = u  
　　　　　　visit (v)  
　　c[u] = black  
　　f[u] = ++time

Q4

計算出的discovery time和finish time如下表所示



產生的forest由5棵樹構成，如下圖。

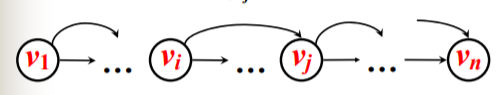


Q5

當一個圖稱作semiconnected，則代表對於所有的u與v，至少存有一條u~>v或v~>u。一開始的想法為一個圖中可能有環可能存在scc，而對scc來說其中任兩點u與v皆有u~>v且v~>u，符合semiconnected的規則，因此有以下的作法。

(1)先找出G所有的scc並將所有的components視為vertices，這些vertices所形成的圖為G’為一DAG。

(2)再對G’做topological sort 並檢查是否形成linear chain也就是對於所有i ，v[i]與v[i+1]是否皆存在edge，如果皆存在則G為semiconnected，如果有任相鄰兩點不存在邊則G不為semiconnected。



Proof: 對scc來說其中任兩點u與v皆有u~>v且v~>u，符合semiconnected的規則，所以如果以components形成的vertices可形成linear chain，也就代表各個scc中的vertices與另一個scc中的vertices有一條路。

時間複雜度：找scc 可以利用dfs為O(|V|+|E|)，而topological sort也是為O(|V|+|E|)，而檢查是否為linear chain為O(|V|)，因此時間複雜度為O(|V|+|E|)。

Q6

該 DFS 實作可以用來解 Topological Sorting。如果在有向圖當中其輸出便是其完成的先後。所以如果再加上一個表格來記錄所有 vertex 的 indegree，便可以拿來做 Topological sorting。

Vertex\_next\_modified ( )  
　　u = stack.pop( )  
　　for each v ∈ Adj[u]  
　　　　if v haven’t been visited  
　　　　indegree\_of\_v --  
　　　　if indegree\_of\_v == 0  
　　　　　　mark v as visited  
　　　　　　push v into stack  
　　return u

分析：

在建立各個 vertex 的 indegree 表格與執行 sorting 時皆需要把每個 vertex 跟 edge 都看過，所以時間複雜度為 。

Q7

想法：If an orientation exist , then 沒有bridge 存在

用Low去找關節點，如果Low[v] ==d[v] 則v是關節點，如果關節點有邊相連則該條邊是bridge

DFS\_Low(u){ //找該點的Low  
　　Low[u] = d[u] = ++time ;  
　　for each v in Adj[v] {  
　　　　if (v haven’t been seen yet) {  
　　　　　　pred[v] = u ;  
　　　　　　DFS\_Low(v);  
　　　　　　Low[u] = min(Low[u],Low[v]);  
　　　　}else if (v != pred[u]) {  
　　　　　　Low[u] = min(Low[u],d[v]);  
　　　　}  
　　}  
}

Find Bridge {   
　　for each v in V {  
　　　　u = pred[v] ;  
　　　　if u != Nil and d[v] == Low[v] {  
　　　　　　edge (u,v) is bridge ;  
　　　　}  
　　}  
}