HW2

110550085 房天越

Part1:

Implementation Details:

先進行一次 DFS，同時進行 Topological Sort 收集每個頂點的 finish，也判斷是否有環，若有則將 IsAcyclic 設為 false，開始找 SCC，若無則繼續進行 Topological Sort。

Topological Sort:

Step1: 建立鄰接矩陣 graph。

Step2: 做 DFS，若途中遇到灰色的(visit=1)的頂點，表示有環，需要做 SCC，將 isAcyclic 標為 false，若遇到黑色的(visit=2)的頂點，不做事

直接 return。

對每一點做迴圈，若該點與給定點有連接且未被拜訪，則以該點繼續做 DFS。

Step3: 以時間 time 記錄離開的順序 order，每當一個頂點被標為黑色，time 就+1(跟 timeforSCC 不同)。

Step4: 若最終 isAcyclic 仍為 0，把 order 由 index 大到小輸出，即是 Topological Sort。

若 isAcyclic=false，則初始化 visit。

Strongly Connected Component:

Step1: 建立轉置鄰接矩陣 rev_graph。

Step2: 做上面的 Topological Sort 的 DFS 同時，以 timeforSCC 紀錄每一點的離開時刻 finish，每當一點變色 timeforSCC 就+1。

Step3: 由小到大排序 finish，存入 finishOrder，最後用 reverse 轉成由大到小。

Step4: 以 finishOrder 的順序對 rev_graph 做第二次 DFS，取得每一頂點的 predecessor。

Step5: 檢查每一點的 predecessor，若=-1，則 SCC 的數量 num_cc+1，填入 scc 陣列，接著對每一點檢查，若檢查點的 predecessor=該點，則填入該點的 scc 陣列。

Step6: 將 SCC 視為一點，做出粗化矩陣 coarseGraph，計算每兩個 SCC 之間　　邊的數量，填入粗化矩陣。

Step7: 計算有多少組 SCC 之間有邊，若有則 line++。

Step8: 輸出 num_cc、line。

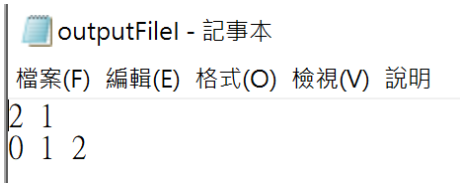
Step9: 對粗化矩陣中每一組，輸出出發點、終點、以及其上邊的數量。

Results:

偵錯主控台畫面:

Output1:

Output 2:



Discussion:

1.Time complexity:

Topological Sort:一次 DFS， O(n+m)。

Strongly Connected Component: 整個算法中最久的是排序 finishOrder，O(n*m)。

2. Challenges that I encountered:

SCC 的找法很難理解，需要兩次 DFS 的原因更是花了半天看了好幾遍文章，另外題目的輸出也花了一段時間理解。

Part2:

Implementation Details:

Dijkstra's Algorithm:

Step1: 建立鄰接矩陣，每一格設成無限大(999999)，若 i==j 則設為 0。

Step2: 建立 visit 陣列，初始化為全部 false。

Step3: 讀入每一條邊的起點、終點和權重，填入鄰接矩陣。

Step4: 從 0 點開始，找出最近的點，若找不到表示已找完最短路徑，Break 離開迴圈，若找得到，則對於每一個未拜訪過的相鄰頂點，若該頂點上的目前 distance 大於目前頂點的 distance+到該點的 Distance，則更新該頂點 distance 為後者。

Step5: 紀錄 Dijkstra 的答案為 dans，以待之後輸出。

將 distance 初始化，準備做 Bellman-Ford's Algorithm。

Bellman-Ford's Algorithm:

Step1: 建立 edgelist 存放所有邊的出發點、終點和權重。

Step2: 進行最多 n-1 次(n 是頂點個數)的迴圈：
對每條邊跑一次：
若出發點 distance 不是無限大，且出發點 distance 加上邊的權重小於終點目前的權重，則更新終點的權重為前者。
另設定一 check 值，感應當前的 distance 是否有更新，若跑完一次大迴圈仍未更新，表示之後也不會更新，就 break 以節省時間

Step3: 檢查是否有 Negative loop，若跑完以上迴圈，再跑一次仍有 Relax，表示有 Negative loop，將 negativeloop 設為 true。

Step4: 紀錄 Bellman-Ford 的答案為 bans。

最後輸出 dans 和 bans 到檔案中，若 negativeloop 為 true 則不輸出 bans，而輸出
"Negative loop detected!"。

Result:

Output3:

Discussion:

1. Way to detect Negative loop in Bellman-Ford's Algorithm:

在跑完 n-1 次對所有邊嘗試 relax 的迴圈後，若沒有負迴圈存在，則應該對每個頂點都已經找到從 0 點到其的最小 distance，但若有負迴圈存在，每跑一次，distance 就必減少，而沒有跑完的一天。於是可知，若跑完 n-1 次迴圈後，仍檢測到 distance 減少，有 relax 發生，則圖中必存在 Negative loop。

2. What to do if the path is required to be printed out:

程式裡面寫一個 parent 陣列，在每次更新的時候，用 parent 陣列紀錄該頂點的前一個頂點，最後建立一個 stack，把這些 parent 從最後一個開始一個一個丟進去，再一個一個印出來。

3. Compare the time complexity of the two algorithms:

Dijkstra: O(n^2)

Bellman-Ford: O(nm)

由於邊數通常會比頂點多，因此 Dijkstra 通常比 Bellman-Ford 快，但 Dijkstra 無法偵測負迴圈，是 Bellman-Ford 不可或缺的理由。

若在不會有負邊的情況下，通常 Dijkstra 會是比較好的選擇。