一、執行結果

User@LEO MINGW64 /d/Users/User/Desktop/DS_hw7
\$ gcc -std=c11 ./*.c -o hw7

User@LEO MINGW64 /d/Users/User/Desktop/DS_hw7
\$./hw7.exe < input0_windows.txt > ans_output0_windows.txt

User@LEO MINGW64 /d/Users/User/Desktop/DS_hw7
\$ diff ./output0_windows.txt ./ans_output0_windows.txt

二、流程圖

讀取 input 並將其存入陣列中



從 ending vertex 中找出圖的終點,並將其存至 last vertex 中(之後宣告 陣列時才知道要多大)





建立 early 的 list,並計算每個 vertex 的 count



依照規則(下面描述)計算出每個 vertex 的 early 值



由終點的 early 可得 critical length



建立 last 的 list,並計算每個 vertex 的 count



依照規則(下面描述)計算出每個 vertex 的 late 值



每個 edge 的 early 就是它 starting vertex 的 early, late 則是 ending vertex 的 late 減去該 edge 的 weight



若 edge 的 early 等於 late,則該 edge 在 critical path 上,要將其印出

三、函式說明

- void InsertList(struct List **first, struct List **cur, int _point, int _weight) 建立 early/ late 的 adjacency list。對 early 而言,_point 傳的是目前處理的 vertex 接到的 vertices,_weight 是這些 vertices 的 weight,count 則代表有幾個 vertices 接到目前處理的 vertex;對 late 而言,_point 傳的是接到目前處理的 vertex 的 vertices,_weight 是這些 vertices 的 weight,count 則代表目前處理的 vertex 接到幾個 vertices。
- void TraverseList(int (*count)[], struct List *first, int outFromStack, int (*state)[], struct Stack **top, int whichState)
 根據 whichState 等於 early 或 late 而做不同的計算
 ◎early
 - 1. 先將所有 vertices 的 early 歸零,並將起點壓入 stack 中。
 - 2. 將在 stack 的 top 的 vertex 彈出來,將此 vertex 的 early 加上它連到的 vertices 的 weight,若結果大於那些 verticies 現有的 early,則將那些 vertices 的 early 更新成該結果。

 - 4. 回到步驟 2,直到 stack 空了為止。

Olate

- 1. 先將所有 vertices 的 early 設為 critical length,並將終點壓入 stack 中。
- 2. 將在 stack 的 top 的 vertex 彈出來,將此 vertex 的 late 減去連到它的 vertices 的 weight,若結果小於那些 vertices 現有的 late,則將那些 vertices 的 late 更新成該結果。
- 3. 將連到它的那些 vertices 的 count 減 1,將 count 等於 0 的 vertices 壓入 stack。
- 4. 回到步驟 2, 直到 stack 空了為止。