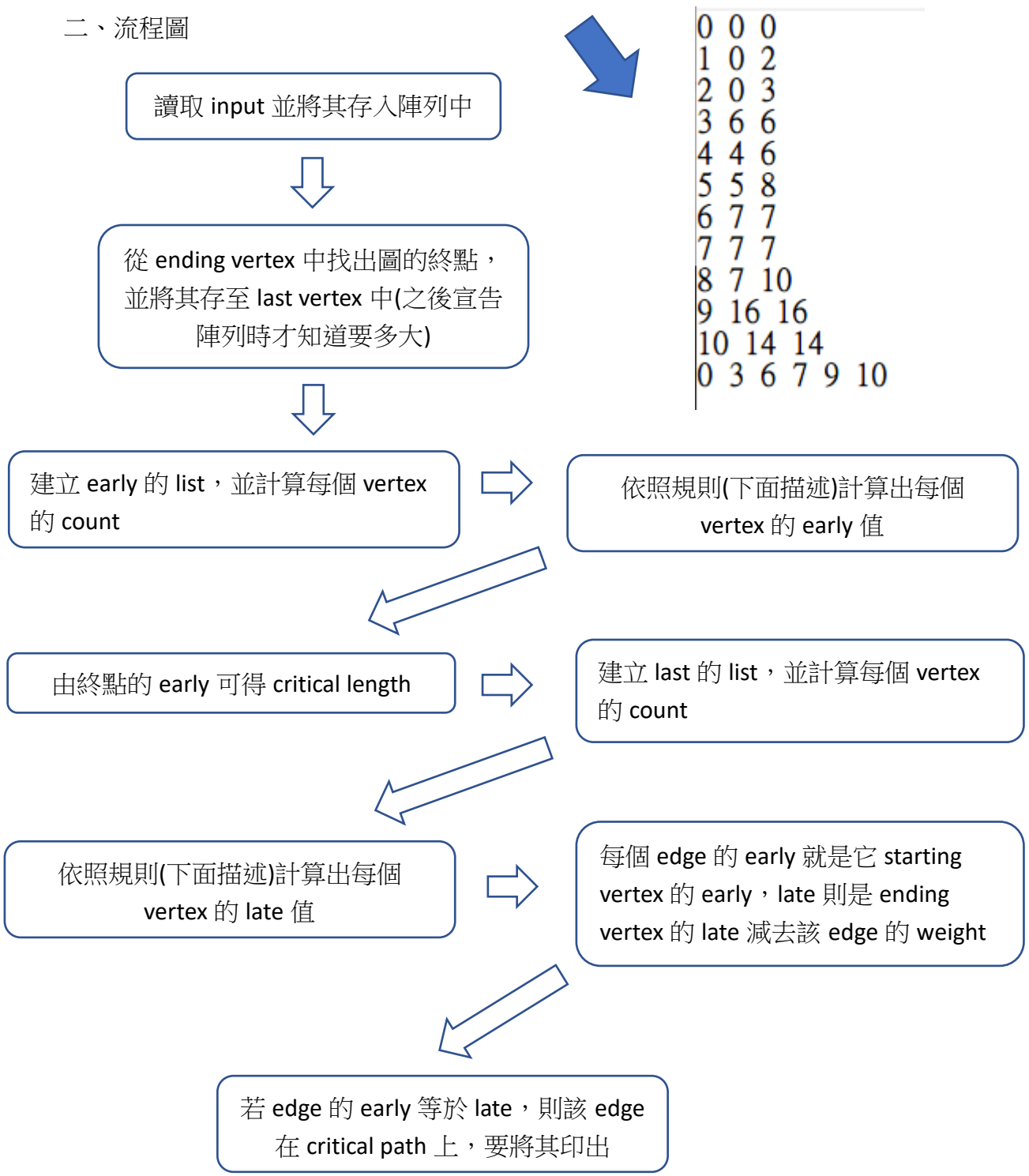
一、執行結果

```
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



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。
  3. 將它連到的那些 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。

  ◎late
  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 verticies 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。