Result Screenshot

Program Architecture

1. 讀入資料並按照 adjacency list 的方式儲存
2. 把每條 path 都按照輸入的資料建好
3. 利用自訂函式算出每個 node 之 early 與 late
4. 用 node 之 early,late 與線段之 weight 算出每條路的 early 和 late
5. 用自訂函式來輸出
6. 利用自訂函式順便找出 critical path

Program Function

void stkpsh(Struct_of_stack**,int);

這是用來處理索引值堆疊 push 的函式，內容寫法是參考之前資結講義上面的程式碼與網路上的教學

int stkpop(Struct_of_stack**);

這是用來處理索引值取出的函式，內容寫法是參考之前資結講義上面的程式碼與網路上的教學

void add_to_link(Link*,Link*,int,int);

這是用來將 link 新添一些新的 path 的函式，其功能也可在 early 與 late 建表時使用

void itl(Link*,Link*);

這是用來將 struct 的內容先初始化的函式，免得有殘存值

void show(Link*,int);

這是用來 debug 的函式，印出一些內容來確認程式目前的運行是正確的，不過後來 debug 完後沒有呼叫了

void early_fetcher(Link*,int*,int);

這是用來將 early 找出來的函式，先將初始的 node push 進去堆疊裡面，然後由堆疊的前端抽取 node，以該 node 為主，去搜尋拓展所有的 path，並更新 early 表格裡面的對應值

void late_fetcher(Link*,int*,int);

上面的函式是由左往右，這個函式變成由右往左，不同的地方還有:這個函式是更新 late 表格內的值，其餘的部分皆與 early_fecture 一樣

void printout(Input*,int*,int*,int);

這是用來印出這次題目所要的結果的函式，裡面也有包含找出 critical path 的功能。

設計過程與思路

這次的作業主體就是圍繞上述的幾個副程式進行運作的，我是從 HW6 的內容直接進行添加與修改，所以內部有些上次的東西，不過我把它們都弄成不會影響這次程式結果運行的狀態了，這次程式裡面含有一些為了 debug 而自訂的變數或是副程式，由於時間緊迫的關係，我沒有將其一一註解或是刪除，因此以不影響程式輸出結果為原則，他們還是留在了我的程式碼內，因此程式看起來有點亂。

這次設計了一些 structure 分別是用來處理節點編號、路徑、節點的下個節點與通往該節點路徑的權重、用來處理抽取的堆疊、用來處理輸入的結構等等。