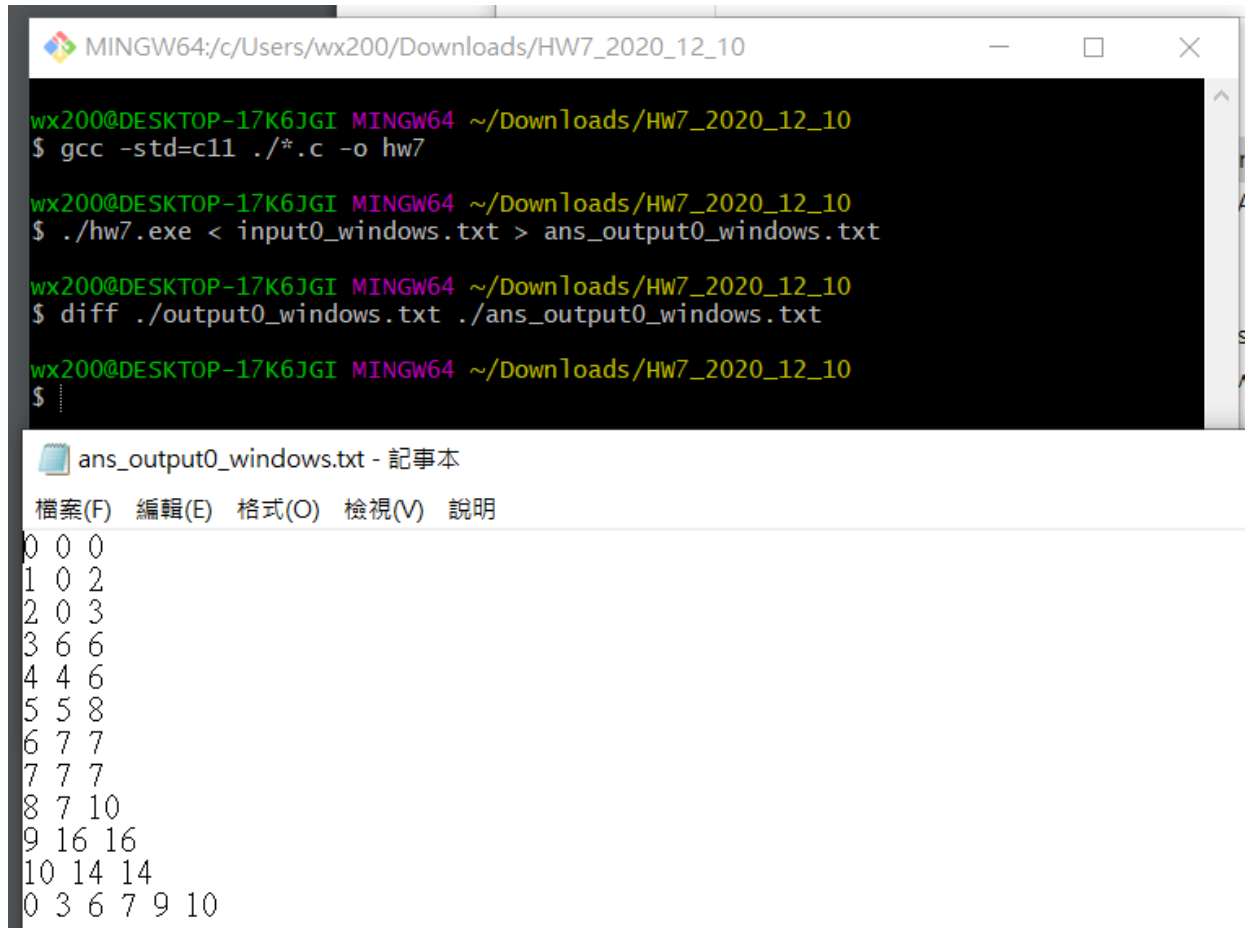


Result screen shot



The screenshot shows a MINGW64 terminal window with the following commands and output:

```
MINGW64:/c/Users/wx200/Downloads/HW7_2020_12_10
wx200@DESKTOP-17K6JGI MINGW64 ~/Downloads/HW7_2020_12_10
$ gcc -std=c11 ./*.c -o hw7
wx200@DESKTOP-17K6JGI MINGW64 ~/Downloads/HW7_2020_12_10
$ ./hw7.exe < input0_windows.txt > ans_output0_windows.txt
wx200@DESKTOP-17K6JGI MINGW64 ~/Downloads/HW7_2020_12_10
$ diff ./output0_windows.txt ./ans_output0_windows.txt
wx200@DESKTOP-17K6JGI MINGW64 ~/Downloads/HW7_2020_12_10
$
```

Below the terminal, a Notepad++ window titled "ans_output0_windows.txt - 記事本" displays the output of the program:

```
檔案(F) 編輯(E) 格式(O) 檢視(V) 說明
0 0 0
1 0 2
2 0 3
3 6 6
4 4 6
5 5 8
6 7 7
7 7 7
8 7 10
9 16 16
10 14 14
0 3 6 7 9 10
```

Program structure

這次的程式結構分了幾個部分

1. 決定資料儲存的方式 (用教授的方式 adjacency list)
2. 處理輸入每條路的部分
3. 開始找出每個節點的 early 和 late
4. 用節點的 early 和 late 與路的權重來算出路的 early 和 late

5. 處理輸出的部分，並順便找出 critical path

而這次使用的 structure 總共有 edge、link、stack 和 input

Link：用來儲存每個節點的編號和各自擁有的路徑

Edge：用來給 Link 接的，

儲存這個節點中各個路所通往的節點和權重，

Stack：用來輔佐找尋 early 和 late 中的抽節點過程（堆疊）

Input：用來輔佐輸入的部分，儲存了每條路的資訊

Program Function

這次 function 還蠻多的

Initial_link() :

用來輔佐 link 結構的初始化

每個節點一開始還沒有任何資訊，需要將資料都清乾淨

Link_Add() :

用來輔佐 Link 去增加新的路

可以用來分別給 Early 和 late 建表的時候使用

Find_Early() :

用來模擬找尋 **Early()** 的過程

總之就是先將初始節點加入 **stack**

接下來一直從 **stack** 中抽出一個節點

再從該節點去擴展所有的路，更新 **early** 表的值

並根據情況加入新的節點到 **stack**

重複上述動作，直到 **stack** 被抽光

Find_late() :

同上述，只是更新的是 **late** 的表

以及路是從右往左擴展

Stack_push() 、 Stack_pop() :

分別是模擬 **Stack** 的 **push** 和 **pop**

使用的是之前第三章所教的堆疊塞入與抽出的方法

Output() :

在找出每個節點的 **Early** 和 **late** 後

接下來就是要算出每條路的 **Early** 與 **late**

最後找出 Critical path

而 Output()就是在做這件事，同時也順便輸出答案

找 Critical path 的方式就是看路的 Early 與 late 是否相等