

# Part1

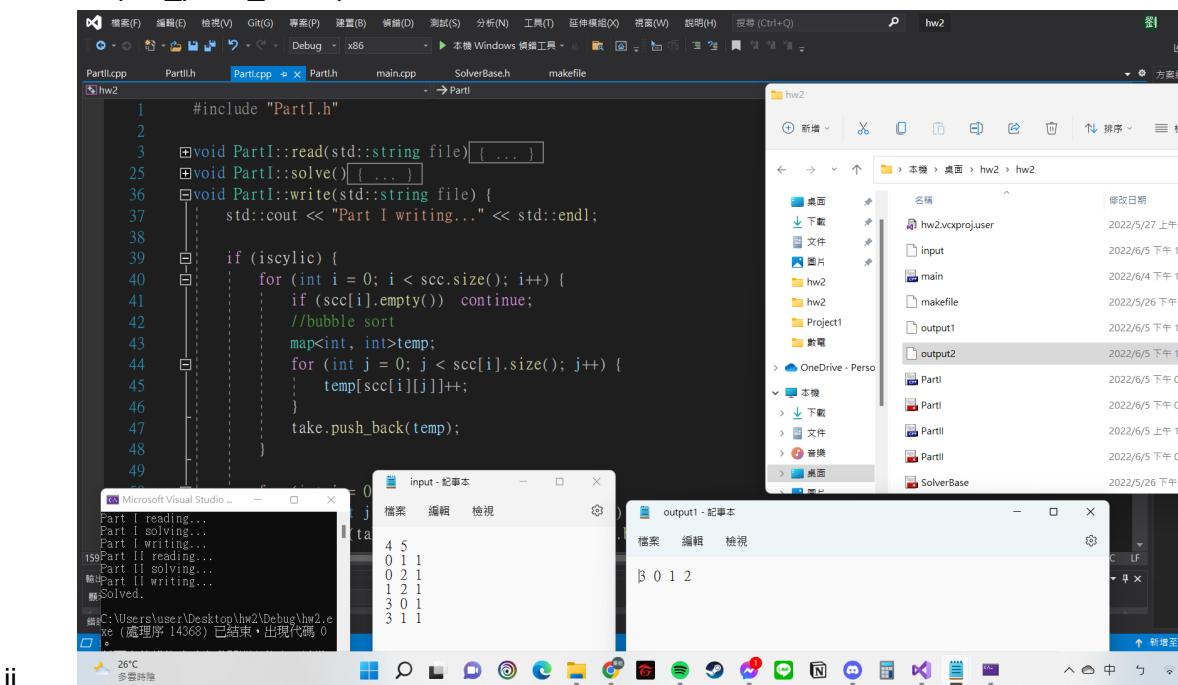
## Step

### 1. Read

- 用`vector<list<pair<int, int>>> graph`裝`input.txt`
- 用同上格式裝"逆graph", edge相同、目的地和來源地相反
- 用`vector<map<int, int>> finish`裝`input.txt`, 方便kosaraju輸出時尋找edge

### 2. Topological sort

- DAG
  - 先用`isThereACycle() == 0`判斷有沒有cycle, dag才能進Topo sort
- 進行Topo sort
  - 初始化`to_graph`, 每個頂點的目的地, 由大到小排序、存入
  - 用迴圈進行dfs, vector `visit`紀錄是否走過, stack `topo_order`存dfs逆序
- 答案
  - 將stack `topo_order.top`出來, 從最後traverse的值印出
- 截圖
  - 用`spec_part1_example1`



ii.

### 3. Kosaraju

- 進行kosaraju
  - 若`isThereACycle() == 1`, 開始初始化`visit`、`scc`、`path`, `visit`紀錄走過, `scc`紀錄每個頂點屬於的SCC, `path`紀錄離開dfs頂點順序
  - 做兩次dfs, 第二次紀錄頂點的來源地, push進`scc`
- 答案

- i. 最為麻煩，兩層迴圈排序scc族內，由第一個(最小的)排族外順序，計算每個CG到另一個CG的edge weight，並且計算CG weight分為從0 -> 頂點.size -1，和頂點.size -1 -> 0
- ii. 將CG的頂點、edge像input.txt圖格式，存進vector<vector<pair<int, int>>> ans，輸出

### c. 截圖

#### i. 用spec\_part1\_example2

```

1 #include "PartI.h"
2
3 void PartI::read(std::string file) { ... }
4 void PartI::solve() { ... }
5 void PartI::write(std::string file) {
6     std::cout << "Part I writing..." << std::endl;
7
8     if (iscyclic) {
9         for (int i = 0; i < scc.size(); i++) {
10            if (scc[i].empty()) continue;
11            //bubble sort
12            map<int, int> temp;
13            for (int j = 0; j < scc[i].size(); j++) {
14                temp[scc[i][j]]++;
15            }
16            take.push_back(temp);
17        }
18    }
19 }
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49

```

input - 記事本

```

4 8
0 1 1
0 2 1
0 3 1
1 0 1
1 2 1
1 3 1
2 0 1
2 1 1

```

output - 記事本

```

2 1
0 1 2

```

## Challenge & Your discover

### 1. kosaraju output格式

- a. 非常麻煩，找到CG來源地(A)到CG目的地(B)，A每個成員到B沒個成員的edge weight都要計算，一開始read順便用vector<map<int, int>> finish紀錄題目的graph，就是利用search O(1)特性，減少時間複雜度

### 2. Topological sort

- a. 要求順序很是麻煩，因為是用遞迴用更難改，我把要topo sort的圖，改成每個頂點的目的地由大到小排好，再讓dfs迴圈由大到小的頂點開始進行，保證選擇到較小的頂點traverse

## Time complexity

### 1. Topological sort

- a.  $O(V+E)$

### 2. Kosaraju

- a.  $O(V+E)$ , 但輸出 $O((VE)^2)$

# Which is better algorithm in which condition

為不同algo, 應無法比較

## Part2

### Step

#### 1. Read

- 將input.txt圖裝入vector<list<pair<int, int>>>, 但分dijkstra用的, 和bellman用的, 前者weight要取正

#### 2. Dijkstra

##### a. 初始化

- vector<int>裝"與起點的距離(distance)"
- priority\_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>>裝"頂點(min\_heap)"

##### b. 進行dijkstra

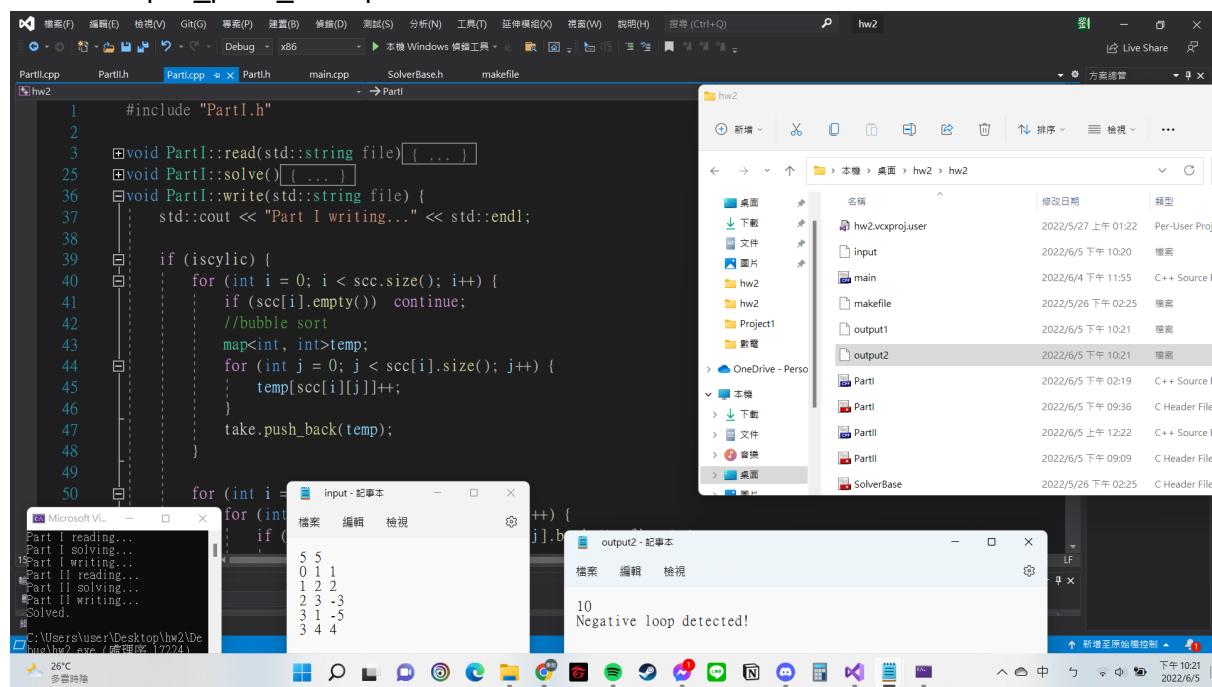
- traverse圖
- 每一次min\_heap.pop都要更新min\_heap數值, 避免有頂點的distance變小, 但沒有優先traverse

##### c. 給答案

- distance結果找到最大值, 存入ans\_di, ans\_di即是答案

##### d. 截圖

- 用spec\_part2\_example3



### 3. Bellman

- a. 初始化
  - i. 同dijkstra
- b. 進行bellman
  - i. 同dijkstra
  - ii. 不同的是，每次尋訪一個頂點前，都要初始化min\_heap，保證用最新版distance
- c. 給答案
  - i. 還要知道第頂點總數次有沒有改distance值，進行最後一次bellman algo。若有改distance值，則有negative cycle，輸出 "Negative...detected!"；若無改distance值，則ans\_be存distance最大值，ans\_be即是答案
- d. 截圖
  - i. 同上

## Challenge & Your discover

### 1. 分class

- a. 原本想再分兩個class，分別給兩個algo，但發現read會很麻煩，只好合在class part1裡面，使得要兩個vector裝兩種algo要用的圖
- b. 但好處是，min\_heap和distance更新和初始化可以用同一個function，寫一個就好

### 2. 裝頂點的heap

- a. 因為heap要有"目的地"和"目的地的distance"，所以用pair，但麻煩是，queue必須依distance排序，本來pair.second=distance，我試圖更改priority\_queue的排序函式，對second排序，但會報錯；於是改成pair.first=distance，似乎visual studio自動判定第一個int，即可動作

## Time complexity

### 1. Dijkstra

- a.  $O(V^2)$

### 2. Bellman

- a.  $O(VE)$

## Which is better algorithm in which condition

### 1. Dijkstra

- a. 當沒有negative weight時候
- b. 應是edge較頂點多的時候

### 2. Bellman

- a. 應是頂點較edge多的時候