109 學年度第二學期資訊期末專題報告

-------以圖論解決生活中可能遇見的問題

作者：柯尹萱
指導老師：陳怡芬
日期：2021/06/16

前言

生活中有很多問題其實都可以用圖論解決，例如：尋找最短路徑、網路流.....等。這份報告主要使用的是圖論中的關節點問題，並和同班同學—邱梓瑄所做的一筆畫問題共享資源，而此份報告中的情境並非真實生活中的情境。

一、動機及目的

圖論對於我們的生活影響極大，多數人常使用的 Google Map 便是利用了最短路徑的相關概念，為使用者提供花費最少時間的路線。除了最短路徑外，生活中的網路流或疾病分析也常使用圖論來解決。大多數人在做圖論問題時，都是以最短路徑、一筆畫等為主題。例如：捷運從 a 棧道 b 站須經過多少時間、郵差送信是否能不重複走任何一條路就把信送完，或著是尋找最想成本的最小生成樹相關問題、最大流最小割定理等。

而我的報告住題主要是想透過尋找關節點的方法找出一張連通的地圖的連接點，並判斷當點被拿掉後會對原地圖造成多大的影響。

二、圖論簡介

圖論是組合數學中一個很重要的分支，起源於著名的柯尼斯堡七橋問題。圖論主要是在探討「圖」，通常是要找圖中的點和邊的關係。圖論和數學中的拓樸學、群論、矩陣論都有密切的關係。

三、問題背景介紹

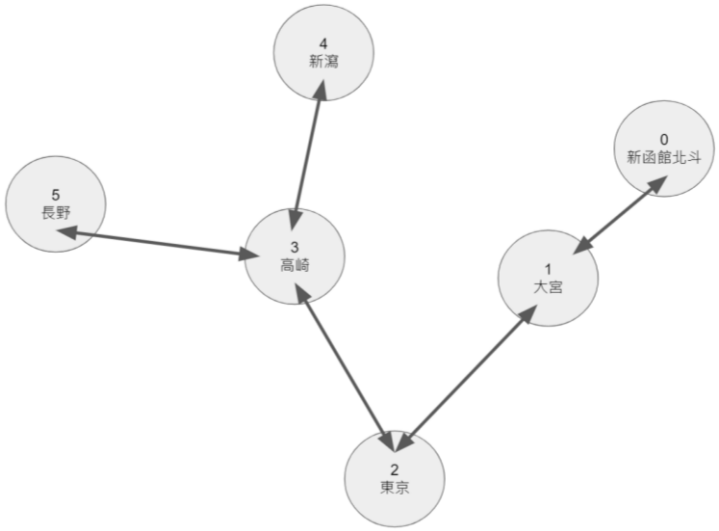
關節點問題主要是透過 DFS 檢查在一張所有點都連通的地圖中，當一個點被拿走的時候，是否會將地圖由原本的一個圖塊分為多個圖塊，進而造成連通圖被拆散。我做這個主題主要是想看看在世界重要的航線地圖中，哪個地區被封住時會對整個航線地圖產生影響。同時也觀察在日本新幹線的地圖中，哪一個車站會使得新幹線路線圖被分隔成兩個以上不連通的區塊。

（圖一）日本新幹線路線圖

四、資料儲存與時間複雜度

Ⅰ 資料儲存

用 vector 去實作 adjacency list，也就是用 vector 來記錄每一個點連到的邊。在 a 的 vector 中若有儲存 b，則代表 a 可以走到 b，而由於我的問題背景為無向圖，因此在 b 的 vector 中也會儲存 a。



（圖二）範例：日本新幹線局部圖

（圖二）名稱對照表：

0：新函館北斗
1：大宮
2：東京
3：高崎
4：新瀉
5：長野

（圖二）的 adjacency list：

0：1
1：0、2
2：1、3
3：2、4、5
4：3
5：3

Ⅱ 時間複雜度

令 V 為地圖上有幾個點，E 為地圖上有幾條邊。

countComponentNum( )每次執行會呼叫若干次 dfs( )，每個點和每個邊會跑到恰好一次，

合計的時間複雜度為 V+E。而我的程式碼中 countComponentNum( )總共被呼叫 V 次，因此時間複雜度為 O(V(V+E))。

五、完整程式碼

程式碼網址：https://replit.com/@d10831415/finalproject15#main.cpp

完整程式碼：

```
#include<iostream>
#include<vector>
using namespace std ;

constexpr int V_MAX = 205 ;

vector<int> edges[V_MAX] ;
bool visited[V_MAX] ;
int V, E ;
int component_num ; // connected component
string place_name[V_MAX] ;

void dfs(const int& cur, const int& ban_v){
  if(visited[cur] || cur==ban_v) return ;
  visited[cur] = true ;
  for(int i=0 ; i<edges[cur].size() ; i++){
    dfs(edges[cur][i], ban_v) ;
  }
}

int countComponentNum(const int ban_v=-1){
  for(int v=0 ; v<V ; v++) visited[v] = false ;
  component_num = 0 ;
  for(int v=0 ; v<V ; v++){
    if(ban_v==v || visited[v]) continue ;
    dfs(v, ban_v) ;
    component_num++ ;
  }
  return component_num ;
}

int main(){
```

```
  cout << "Please input the number of vertixes and edges:" << endl ;
  cout << "-----------------------------------------------------------"
<< endl ;

  while(cin>>V>>E){
    cout << endl << "Please input the name of the places(in the order
of the numbers):" << endl ;
    cin.get() ;
    for(int v=0 ; v<V ; v++){
      getline(cin, place_name[v]) ;
      edges[v].clear() ;
    }
    cout << "-----------------------------------------------------------"
<< endl ;
    cout << "Please input the map using numbers and the order of the places
you input, please start form 0: " << endl ;
    int a, b ;
    for(int e=0 ; e<E ; e++){
      cin>>a>>b ;
      edges[a].push_back(b) ;
      edges[b].push_back(a) ;
    }
    // finding articulation vertex
    if(countComponentNum()>1){
      cout << "This graph is not connected." << endl ;
      continue ;
    }
    bool a_point_found = false ;
    for(int v=0 ; v<V ; v++){
      int component_num = countComponentNum(v) ;
      if(component_num>1){
        cout << place_name[v] << " is an articulation point. It separates
the graph into " << component_num << " components." << endl ;
        a_point_found = true ;
      }
    }
    if(!a_point_found) cout << "There is no articulation point in this
graph." << endl ;
```

```
  }
  return 0 ;
}
```

六、程式碼介紹

```
constexpr int V_MAX = 205 ;

vector<int> edges[V_MAX] ;
bool visited[V_MAX] ;
int V, E ;
int component_num ; // connected component
string place_name[V_MAX] ;
```

我設定這支程式碼的地圖最多可以有 200 個點，由於這個數值不會改變，我設了一個 constexpr 的變數來儲存。兩點之間的連線在這邊是用一維 vector 來處理，而判斷是否有走過則以 bool 型態的 visited 儲存。V 和 E 分別表示點的數量以及邊的數量，component_num 是用來記錄地圖上現在有幾個區塊，而字串型態的 place_name 則是用來記錄點的名稱。

```
int main(){

  cout << "Please input the number of vertixes and edges:" << endl ;
  cout << "-----------------------------------------------------------" <<
endl ;
```

執行程式時，先以文字訊息告知執行者輸入地圖上點和邊的數量。

```
  while(cin>>V>>E){
    cout << endl << "Please input the name of the places(in the order of the
numbers):" << endl ;
    cin.get() ;
    for(int v=0 ; v<V ; v++){
      getline(cin, place_name[v]) ;
      edges[v].clear() ;
    }
    cout << "-----------------------------------------------------------" <<
endl ;
```

一開始先輸入地圖上點以及邊的數量，接著輸出文字訊息告知使使用者輸入各個點的名稱，這邊的輸入不使用 cin 使用 getline( )的原因是因為地名有時候會有空格，所以為了避免 cin 忽略空格，就改成使用可以一次把整行抓下來的 getline( )。而由於在輸入名稱前會先輸出一段文字，為了避免文字最後的換航段程式執行造成影響，在開始輸入前顯已 cin.get( )將換行拿掉。輸入名稱的過程中，同時也把儲存路徑的 edges 清空。

```
    cout << "Please input the map using numbers and the order of the places
you input, please start form 0: " << endl ;
    int a, b ;
    for(int e=0 ; e<E ; e++){
      cin>>a>>b ;
      edges[a].push_back(b) ;
      edges[b].push_back(a) ;
    }
```

輸出文字訊息告知使用者在輸入地圖的路徑時，需要依照上面名稱的順序進行編號，並且從 0 開始。開兩個暫存路徑起始跟終點的整數變數 a、b，輸入時因為地圖是無向圖，所以 edges 的 a 跟 b 兩格都要記得存。

```
    // finding articulation vertex
    if(countComponentNum()>1){
      cout << "This graph is not connected." << endl ;
      continue ;
    }
```

這邊在開始找關節點前，要先判斷原本的地圖是不是只有一個區塊如果 countComponentNum( )這個函數回傳的值大於 1，那就輸出 This graph is not connected.來告知使用者這張地圖有超過一個圖塊，那也就沒有所謂的關節點。

```
int countComponentNum(const int ban_v=-1){
  for(int v=0 ; v<V ; v++) visited[v] = false ;
  component_num = 0 ;
  for(int v=0 ; v<V ; v++){
    if(ban_v==v || visited[v]) continue ;
    dfs(v, ban_v) ;
    component_num++ ;
  }
  return component_num ;
}
```

countComponentNum( )這個函式主要是用來判斷地圖上現在總共有幾個圖塊，它會回傳一個整數值，因此它的型態是 int。呼叫這個函式時，需要給他一個整數直 ban_v，這個整數值代表的是我現在要拔掉的點。若我沒有傳任何數值給它，那麼 ban_v 會自動用-1 代入。進到 countComponentNum( )後，第一件要做的事就是把所有點的 visited 都改成 false，因為這樣面在 dfs 的時候才不會出問題。接著將計算圖塊數量的變數 component_num 定為 0，並開始計算數量。找關節點主要就是要看每一個點有沒有相連，並且這些相連的點是否剛好構成一幅地圖。因此我對每一個點都做 dfs，以便判斷地圖否連通。而在進行 dfs( )前，需要先判斷現在看的這個點是否和 ban_v 為同一點，弱勢，則直接 continue 往下看，原因是 ban_v 是我

要嘗試拿掉的點，不能納入判斷。而另一個不需要判斷的是已經走過的點，若這個點的 visited 等於 true，就直接 continue。將不合理的點都判斷完後，就可以開始 dfs( )了。而因為一次 dfs( )代表走完一個區塊，因此每做完一次 dfs( )，countComponentNum( )就要加一。在走完每一個點後，回傳 component_num。

```
void dfs(const int& cur, const int& ban_v){
  if(visited[cur] || cur==ban_v) return ;
  visited[cur] = true ;
  for(int i=0 ; i<edges[cur].size() ; i++){
    dfs(edges[cur][i], ban_v) ;
  }
}
```

dfs( )這個函式顧名思義就是在做 dfs( )做的事。要執行 dfs( )這個函式需要傳入兩個整數值，分別是現在所在的點 cur 以及要拔掉的點 ban_v。進到 dfs( )後首先要判斷現在在地點是否已經被走過和這個點是否就是 ban_v，若符合上述任一個條件，則直接離開函式。若否則將 cur 的 visited 改成 true，並且對 cur 連接到的點做 dfs( )。

```
    bool a_point_found = false ;
    for(int v=0 ; v<V ; v++){
      int component_num = countComponentNum(v) ;
      if(component_num>1){
        cout << place_name[v] << " is an articulation point. It separates
the graph into " << component_num << " components." << endl ;
        a_point_found = true ;
      }
    }
    if(!a_point_found) cout << "There is no articulation point in this
graph." << endl ;
  }
  return 0 ;
}
```

在判斷完地圖是否只有一個區塊後，接著就是要開始找關節點了。找關節點的方法就是把每一個點都拿掉並且再做一次 dfs，也就代表在這個 for 迴圈裡面呼叫 countComponentNum( )時，傳入的 ban_v 會是點的編號。而當 countComponentNum( )回傳的值大於 1 得時候，就輸出這個點的名字以及將這個點拔掉會把整個地圖分成幾個圖塊。離開 for 迴圈後，如果都沒有找到關節點，則就要輸出這張地圖上沒有關節點的文字。為了方便判斷，在這裡我開了一個 bool 變數 a_point_found，只要過程中輸出一次關節點，就把 a_point_found 改為 true，在離開 for 迴圈後以 if 判斷是判斷是否有找到。

七、實際測試結果

範例測資 1：

```
31 55
Far East
West Coast of North America
East Coast of North America
Caribbean
West Coast of South America
Indian
South-East Asia
Australia
New Zealand
East Africa
Mediterranean Sea
North-west Europe
Cape of Good Hope
West Africa
North of Indian Ocean
Asia Pacific
Europe
Persian Gulf
West Europe
Suez Canal
Gibraltar
Indian Ocean
Central South Africa
East Coast of South America
Japan
California
Seattle
Vancouver
New York
United Kingdom
Great Lakes
0 1
0 2
0 3
0 4
0 5
0 6
```

0 7
0 8
0 10
0 16
2 3
2 20
6 9
6 10
6 12
6 24
8 1
8 2
10 1
10 2
10 11
10 18
10 19
10 23
11 2
11 3
11 13
11 23
12 0
12 4
12 13
12 18
12 23
13 4
13 21
13 22
14 15
14 16
16 2
16 7
17 6
17 12
17 19
19 10
19 21

```
20 10
21 15
22 12
23 12
24 25
24 26
24 27
24 28
29 2
30 16
```

範例結果 1：

```
Far East is an articulation point. It separates the graph into 2 components.
East Coast of North America is an articulation point. It separates the graph into 2 components.
South-East Asia is an articulation point. It separates the graph into 3 components.
Europe is an articulation point. It separates the graph into 2 components.
Japan is an articulation point. It separates the graph into 5 components.
```

由第一筆測資的結果可以看到這張地圖有 5 個關節點，分別是 Far East、East Coast of North America、South-East Asia、Europe 和 Japan。這五個地區任一個被拿掉後，整個航線地圖就會有點無法和其他地區連通。

範例測資 2：

```
115 114
Shin-Hakodate-Hokuto
Kikonai
Yunosato-Shiriuchi
Yoshioka-Teiten
Tappi-Teiten
Oku-Tsugaru-Imabetsu
Shin-Nakaoguni
Shin-Aomori
Shichinohe-Towada
Hachinohe
Ninohe
Iwate-Numakunai
Morioka
Shin-Hanamaki
Kitakami
```

Mizusawa-Esashi
Ichinoseki
Kurikoma-Kogen
Furukawa
Sendai Eki
Shiroishi-Zao
Fukushima
Koriyama
Shin-Shirakawa
Nasushiobara
Utsunomiya
Oyama
Washinomiya
Omiya
Ueno
Tokyo
Shizukuishi
Tazawako
Kakunodate
Omagari
Akita
Yonezawa
Takahata
Akayu
Kaminoyama-Onsen
Yamagata
Tendo
Sakuranbo-Higashine
Murayama
Oishida
Shinjo
Kumagaya
Honjo-Waseda
Takasaki
Jomo-Kogen
Echigo-Yuzawa
Urasa
Nagaoka
Tsubame-Sanjo

Niigata
Annaka-Haruna
Karuizawa
Sakudaira
Ueda
Nagano
Iiyama
Joetsu-Myoko
Itoigawa
Kurobe-Unazuki-Onsen
Toyama
Shin-Takaoka
Kanazawa
Shinagawa
Shin-Yokohama
Odawara
Atami
Mishima
Shin-Fuji
Shizuoka
Kakegawa
Hamamatsu
Toyohashi
Mikawa-Anjo
Nagoya
Gifu-Hashima
Maibara
Ritto
Kyoto
Torikai
Shin-Osaka
Shin-Kobe
Nishi-Akashi
Himeji
Aioi
Okayama
Shin-Kurashiki
Fukuyama
Shin-Onomichi

Mihara
Higashi-Hiroshima
Hiroshima
Shin-Iwakuni
Tokuyama
Shin-Yamaguchi
Asa
Shin-Shimonoseki
Kokura
Kurate
Hakata
Shin-Tosu
Kurume
Chikugo-Funagoya
Shin-Omuta
Shin-Tamana
Kumamoto
Shin-Yatsushiro
Shin-Minamata
Izumi
Sendai
Kagoshima-Chuo
0 1
1 2
2 3
3 4
4 5
5 6
6 7
7 8
8 9
9 10
10 11
11 12
12 13
12 31
13 14
14 15
15 16

16 17
17 18
18 19
19 20
20 21
21 22
21 36
22 23
23 24
24 25
25 26
26 27
27 28
28 29
28 46
29 30
30 67
31 32
32 33
33 34
34 35
36 37
37 38
38 39
39 40
40 41
41 42
42 43
43 44
44 45
46 47
47 48
48 49
49 50
50 51
51 52
52 53
53 54
48 55

55 56
56 57
57 58
58 59
59 60
60 61
61 62
62 63
63 64
64 65
65 66
67 68
68 69
69 70
70 71
71 72
72 73
73 74
74 75
75 76
76 77
77 78
78 79
79 80
80 81
80 82
82 83
83 84
84 85
85 86
86 87
87 88
88 89
89 90
90 91
91 92
92 93
93 94
94 95

```
95 96
96 97
97 98
98 99
99 100
100 101
101 102
102 103
103 104
104 105
105 106
106 107
107 108
108 109
109 110
110 111
111 112
112 113
113 114
```

範例結果 2：

```
Kikonai is an articulation point. It separates the graph into 2 components.
Yunosato-Shiriuchi is an articulation point. It separates the graph into 2 components.
Yoshioka-Teiten is an articulation point. It separates the graph into 2 components.
Tappi-Teiten is an articulation point. It separates the graph into 2 components.
Oku-Tsugaru-Imabetsu is an articulation point. It separates the graph into 2 components.
Shin-Nakaoguni is an articulation point. It separates the graph into 2 components.
Shin-Aomori is an articulation point. It separates the graph into 2 components.
Shichinohe-Towada is an articulation point. It separates the graph into 2 components.
Hachinohe is an articulation point. It separates the graph into 2 components.
Ninohe is an articulation point. It separates the graph into 2 components.
Iwate-Numakunai is an articulation point. It separates the graph into 2 components.
Morioka is an articulation point. It separates the graph into 3 components.
Shin-Hanamaki is an articulation point. It separates the graph into 2 components.
Kitakami is an articulation point. It separates the graph into 2 components.
Mizusawa-Esashi is an articulation point. It separates the graph into 2 components.
Ichinoseki is an articulation point. It separates the graph into 2 components.
Kurikoma-Kogen is an articulation point. It separates the graph into 2 components.
Furukawa is an articulation point. It separates the graph into 2 components.
```

Sendai Eki is an articulation point. It separates the graph into 2 components.
Shiroishi-Zao is an articulation point. It separates the graph into 2 components.
Fukushima is an articulation point. It separates the graph into 3 components.
Koriyama is an articulation point. It separates the graph into 2 components.
Shin-Shirakawa is an articulation point. It separates the graph into 2 components.
Nasushiobara is an articulation point. It separates the graph into 2 components.
Utsunomiya is an articulation point. It separates the graph into 2 components.
Oyama is an articulation point. It separates the graph into 2 components.
Washinomiya is an articulation point. It separates the graph into 2 components.
Omiya is an articulation point. It separates the graph into 3 components.
Ueno is an articulation point. It separates the graph into 2 components.
Tokyo is an articulation point. It separates the graph into 2 components.
Shizukuishi is an articulation point. It separates the graph into 2 components.
Tazawako is an articulation point. It separates the graph into 2 components.
Kakunodate is an articulation point. It separates the graph into 2 components.
Omagari is an articulation point. It separates the graph into 2 components.
Yonezawa is an articulation point. It separates the graph into 2 components.
Takahata is an articulation point. It separates the graph into 2 components.
Akayu is an articulation point. It separates the graph into 2 components.
Kaminoyama-Onsen is an articulation point. It separates the graph into 2 components.
Yamagata is an articulation point. It separates the graph into 2 components.
Tendo is an articulation point. It separates the graph into 2 components.
Sakuranbo-Higashine is an articulation point. It separates the graph into 2 components.
Murayama is an articulation point. It separates the graph into 2 components.
Oishida is an articulation point. It separates the graph into 2 components.
Kumagaya is an articulation point. It separates the graph into 2 components.
Honjo-Waseda is an articulation point. It separates the graph into 2 components.
Takasaki is an articulation point. It separates the graph into 3 components.
Jomo-Kogen is an articulation point. It separates the graph into 2 components.
Echigo-Yuzawa is an articulation point. It separates the graph into 2 components.
Urasa is an articulation point. It separates the graph into 2 components.
Nagaoka is an articulation point. It separates the graph into 2 components.
Tsubame-Sanjo is an articulation point. It separates the graph into 2 components.
Annaka-Haruna is an articulation point. It separates the graph into 2 components.
Karuizawa is an articulation point. It separates the graph into 2 components.
Sakudaira is an articulation point. It separates the graph into 2 components.
Ueda is an articulation point. It separates the graph into 2 components.
Nagano is an articulation point. It separates the graph into 2 components.
Iiyama is an articulation point. It separates the graph into 2 components.

Joetsu-Myoko is an articulation point. It separates the graph into 2 components.
Itoigawa is an articulation point. It separates the graph into 2 components.
Kurobe-Unazuki-Onsen is an articulation point. It separates the graph into 2 components.
Toyama is an articulation point. It separates the graph into 2 components.
Shin-Takaoka is an articulation point. It separates the graph into 2 components.
Shinagawa is an articulation point. It separates the graph into 2 components.
Shin-Yokohama is an articulation point. It separates the graph into 2 components.
Odawara is an articulation point. It separates the graph into 2 components.
Atami is an articulation point. It separates the graph into 2 components.
Mishima is an articulation point. It separates the graph into 2 components.
Shin-Fuji is an articulation point. It separates the graph into 2 components.
Shizuoka is an articulation point. It separates the graph into 2 components.
Kakegawa is an articulation point. It separates the graph into 2 components.
Hamamatsu is an articulation point. It separates the graph into 2 components.
Toyohashi is an articulation point. It separates the graph into 2 components.
Mikawa-Anjo is an articulation point. It separates the graph into 2 components.
Nagoya is an articulation point. It separates the graph into 2 components.
Gifu-Hashima is an articulation point. It separates the graph into 2 components.
Maibara is an articulation point. It separates the graph into 3 components.
Kyoto is an articulation point. It separates the graph into 2 components.
Torikai is an articulation point. It separates the graph into 2 components.
Shin-Osaka is an articulation point. It separates the graph into 2 components.
Shin-Kobe is an articulation point. It separates the graph into 2 components.
Nishi-Akashi is an articulation point. It separates the graph into 2 components.
Himeji is an articulation point. It separates the graph into 2 components.
Aioi is an articulation point. It separates the graph into 2 components.
Okayama is an articulation point. It separates the graph into 2 components.
Shin-Kurashiki is an articulation point. It separates the graph into 2 components.
Fukuyama is an articulation point. It separates the graph into 2 components.
Shin-Onomichi is an articulation point. It separates the graph into 2 components.
Mihara is an articulation point. It separates the graph into 2 components.
Higashi-Hiroshima is an articulation point. It separates the graph into 2 components.
Hiroshima is an articulation point. It separates the graph into 2 components.
Shin-Iwakuni is an articulation point. It separates the graph into 2 components.
Tokuyama is an articulation point. It separates the graph into 2 components.
Shin-Yamaguchi is an articulation point. It separates the graph into 2 components.
Asa is an articulation point. It separates the graph into 2 components.
Shin-Shimonoseki is an articulation point. It separates the graph into 2 components.
Kokura is an articulation point. It separates the graph into 2 components.

```
Kurate is an articulation point. It separates the graph into 2 components.
Hakata is an articulation point. It separates the graph into 2 components.
Shin-Tosu is an articulation point. It separates the graph into 2 components.
Kurume is an articulation point. It separates the graph into 2 components.
Chikugo-Funagoya is an articulation point. It separates the graph into 2 components.
Shin-Omuta is an articulation point. It separates the graph into 2 components.
Shin-Tamana is an articulation point. It separates the graph into 2 components.
Kumamoto is an articulation point. It separates the graph into 2 components.
Shin-Yatsushiro is an articulation point. It separates the graph into 2 components.
Shin-Minamata is an articulation point. It separates the graph into 2 components.
Izumi is an articulation point. It separates the graph into 2 components.
Sendai is an articulation point. It separates the graph into 2 components.
```

由第二筆測資可以看到日本新幹線地圖上，多數點都只會將地圖分為兩個區塊，也就代表在這 108 個關節點車站中，只要其中任一個車站無法讓列車進入，正個新幹線地圖就無法成為一個連通圖。

八、與他人合作

和我同班的邱梓瑄同學在這次報告中做的是一筆畫問題，而我們在經過討論後發現我們可以共享資料，並且程式碼也可以在同一情境中進行結合。例如：假設現在發生一場戰爭，一筆畫問題可以判斷一個國家或聯盟的訊息傳播是否可以一次性傳播給每一個人，而關節點則可以判斷哪些地區的影響較大，進而對這些地區加強防守。抑或是當今天郵差在送信時，一筆畫可以判斷郵差是否可以不重複經過任何地方就把信全部送完，而關節點則可以判斷哪一個路口被封住時郵差受到的影響最大。

九、心得與討論

一開始聽到這次專題的主題時，我還想說會很簡單，因為以前解題有遇到過一些問題背景和真實生活相關的題目。結果沒想到程式碼雖然很快就寫好了，但光是處理資料就弄了超級久。

原本以為海運地圖會很好找，結果沒想到資料少的可憐。好不容易找到了一個網站有，沒想到裡面只有航線，我得自己把地區名稱改成英文在記錄到一旁。而且有些航線連地區都沒給還得要自己查，查完後就要開始把連線用數字存起來。因為是海運地圖，所以很多地區都是交錯出現，要很小心不然就會出現重複輸入或路線彼此蓋過的情況。

而新幹線的地圖更加複雜，原本其實是想做倫敦地鐵圖，但因為倫敦地鐵太雜亂了，就改成新幹線。日本新幹線的地圖看似簡單，但轉運站實在是太多了，一不小心還會重複輸入。而且因為 CodeBlocks 不支援中文，每一個地名都還要自己查拼音再輸入。並且因為有些車站的拼音在中文的網頁上一模一樣，必須上日文網頁看才知道全名，弄了快兩個小時才弄完。

雖然在製作這次專題時，在處理測資上遇到很多困難，包括地名出現不支援的語言或換

行和空格導致資料輸入有問題。但經過這次的訓練我了解到其實用圖論解決問題並不是我們所想的那麼簡單，平常我們只要動動手指就可以得到答案，但其實背後設計的人是花了非常多時間在處理測資以及蒐集資料。

做完這次專題我覺得可以討論的地方是，要如何判斷一個點的影響程度很大。我現在的程式是將所有影響的結果都輸出，但其實每個點造成的影響程度並不一樣，若可以找到一個方法判斷影響程度的大小，就可以避免輸出像新幹線測資一樣有非常多筆的結果。

而另外一個我認為值得探討的問題是有沒有可能將關節點反著做，也就是若現在有一張所有點並不連通的地圖，我們是否可以設計一隻程式碼將現在做的是反過來。變成尋找要在哪裡加入一個點才會使得地圖變成連通圖。

現在所做的關節點是討論哪一個點被抹去時，可以把地圖從一個區塊分隔成多個區塊，但由於無法判斷影響程度大小，解決問題的能力有限。而上述兩個可探討的問題不僅可讓關節點的應用更加廣泛，將其反著做更可以使我們優化現在已有的地圖或訊息網。

十、參考資料

1. Articulation Vertex / Bridge：http://web.ntnu.edu.tw/~algo/Component.html

2. 國際大洋航線：https://wiki.mbalib.com/zh-tw/%E5%9B%BD%E9%99%85%E5%A4%A7%E6%B4%8B%E8%88%AA%E7%BA%BF

3. 新幹線：https://zh.wikipedia.org/wiki/%E6%96%B0%E5%B9%B9%E7%B7%9A

4. 北海道新幹線：https://zh.wikipedia.org/wiki/%E5%8C%97%E6%B5%B7%E9%81%93%E6%96%B0%E5%B9%B9%E7%B7%9A

5. 東北新幹線：https://zh.wikipedia.org/wiki/%E6%9D%B1%E5%8C%97%E6%96%B0%E5%B9%B9%E7%B7%9A

6. 上越新幹線：https://zh.wikipedia.org/wiki/%E4%B8%8A%E8%B6%8A%E6%96%B0%E5%B9%B9%E7%B7%9A

7. 秋田新幹線：https://zh.wikipedia.org/wiki/%E7%A7%8B%E7%94%B0%E6%96%B0%E5%B9%B9%E7%B7%9A

8. 山形新幹線：https://zh.wikipedia.org/wiki/%E5%B1%B1%E5%BD%A2%E6%96%B0%E5%B9%B9%E7%B7%9A

9. 北路新幹線：
https://zh.wikipedia.org/wiki/%E5%8C%97%E9%99%B8%E6%96%B0%E5%B9%B9%E7%B7%9A

10. 山陽新幹線：
https://zh.wikipedia.org/wiki/%E5%B1%B1%E9%99%BD%E6%96%B0%E5%B9%B9%E7%B7%9A

11. 九州新幹線（鹿兒島線）：
https://zh.wikipedia.org/wiki/%E4%B9%9D%E5%B7%9E%E6%96%B0%E5%B9%B9%E7%B7%9A_(%E9%B9%BF%E5%85%92%E5%B3%B6%E7%B7%9A)