

MAY 2021

圖的相異色彩

輸入並判斷色分圖
上色所需的最小顏色數

程式設計實作

Made by : 學生 李沛儒

圖的相異色彩

輸入並判斷 k -分圖上色所需的最小顏色數

零、大綱要點

排列組合為高中數學多而繁雜的課綱內，可說是最變化多端的課程了。高中的教學內容不外乎：重複排列、選取組合、相異路徑、圖的填色……其中填色圖的解法又是如此地五花八門。兩張圖只要相差僅僅一個邊，解出來的答案卻是相去甚遠。那是否有辦法以一個有條理的過程，在遇到同種問題時都能迎刃而解？

程式設計不僅考驗 Coder 的邏輯能力，一串好的程式需要兼顧效率及兼容性，細膩而循序漸進的思維更是一大難題。這次的實作專案將把多變化的題目，藉活用程式設計中的深度優先搜尋(DFS)及圖的演算法，歸納成為一個能自由輸入圖的程式，解決 k -分圖的問題，並探討實作上遇到的問題及討論解決方法。

壹、實作動機

程式設計就是用特定的語法與電腦溝通，並命令它代替人類做一些平時較難處理、資料數很大的文件或資訊。在高中數學排列組合的課程內，關於填色的部分往往只提到「給定一個無向圖並求其塗法數」而已，完全沒有關於 k -分圖的解析說明。的確，因為求取 k -分圖時，面對頂點數較多時確實難以人工完成。但我不能就止於此，我想要深入去研究並了解其運算過程，並結合數學觀念以及程式設計中的深度優先搜尋和圖的演算法以完成本次的目標專案。同時，我會詳細提出我在實作時遇到的問題及如何解決，並思考討論在何些圖形特徵的情況下會無法成功輸出結果。

貳、實作目的

結合數學及資訊科技兩個科目，藉由程式設計解決人工難以計算的數學計數問題，並同時訓練邏輯思考及 coding 能力。

參、先備知識

本次的實作專案是用 Cpp 語法撰寫而成，其中涉及兩個核心的演算法，即為前文所

提到的深度優先搜尋演算法及圖的表示法。以下對兩者進行一些說明：

深度優先搜尋演算法 Depth-First Search, DFS

深度優先搜尋是一種搜尋手法，從某個狀態下開始後，就持續讓狀態遷移，直到無法遷移時便回到前一個狀態，無法得到解答時就回到前狀態並重新決定新的回傳值。通常會以遞迴函數形式寫成。

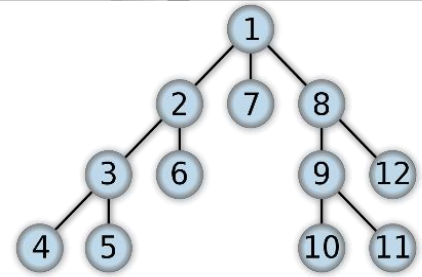


圖 1 DFS 狀態遷移的順序

這個與那個其實都是「圖」 Graph

圖是物件與狀態間之連接方式的表示方法，數個頂點和數條邊可以構成最基本的圖。同時，圖還有分有向及無向圖、連通或非連通圖、多重邊、迴圈、環.....在本篇實作專題可以成功執行的為有向及無向圖、連通或非連通圖、環，並且最後都會對其各自可否的原因進行討論。

肆、實作過程

一、 確認變數

1. 確認並輸入整張圖共有 V 個頂點、 E 條邊。
 2. 需要一個 `vector<int> G[MAX_V]` 的鄰接串列用以記錄圖。
 3. 最後，`color[MAX_V]` 記錄各頂點的顏色。
- (此處僅以「判斷是否為二分圖」為目的撰寫，故先以 1, -1 來代表相異的顏色。)

二、 遍歷頂點

用 `bool dfs` 以 1, -1 來逐步填滿頂點。 v 為當前 `dfs` 執行的頂點， c 為所填的顏色。以 c 填滿頂點 v ，如果相鄰的頂點顏色相同就傳回 `false`，尚未填色的點使用 $-c$ 填，(如圖 2)。

```

1  bool dfs(int v, int c){
2      color[v] = c ;
3      for(int i=0; i<G[v].size(); i++){
4          if(color[G[v][i]] == c) return false ;
5          if(color[G[v][i]] == 0 && !dfs(G[v][i], -c)) return false ;
6      }
7      return true ;
8  }
```

圖 2 DFS 函式的程式碼

三、 主程式撰寫

先輸入有多少頂點 V 和多少邊 E ，再加上頂點 S 到 T 的邊用以建立圖 G ，值得注意的是，因為是無向圖的關係，所以只要在確定 A to B 連通後，直接加上 B to A 就好。

再者是問題並無提到是否為連通圖，所以用 `for` 來走訪各個頂點。正常來說，

是連通圖的情形下只要給定一個初值，藉著 DFS 就可以遍歷所有頂點了，但在遇上非連通圖時，就需要再給定新的初值，找到未填色的頂點時，再次填入 c。(如圖 3)

```

1  int main(){
2      cin >> V >> E ;
3      for(int i=0;i<E;i--){
4          int s, t ;
5          cin >> s >> t ;
6          G[s].push_back(t) ;
7          G[t].push_back(s) ;
8      }
9
10     for(int i=0;i<V;i--){
11         if(color[i]==0){
12             if(!dfs(i, 1)){
13                 cout << "No" << endl ;
14                 return 0 ;
15             }
16         }
17     }
18     cout << "Yes" << endl ;
19     return 0 ;
20 }

```

圖 3 main 函式的程式碼

最後，若為所有頂點都填完色後，相鄰點皆不為同色時則輸出 Yes，反之則 No。

四、 重複結構

試想，如果每次都要重啟程式，等待軟體編譯的時間，是否會相當惱人？

所以我在上述程式確定可以執行後，加入了 while 迴圈來讓程式持續讀取直到 EOF。少了重新編譯的過程，就要面對的是初始化的問題。若沒有初始化的話會造成前後兩張圖疊合，顏色延續前一張圖的災難，所以初始化式重複結構中很重要的一環。

在前版程式(不重複的)，我是用輸出完答案後馬上 return 0 來中止程式，以確保不會輸出到兩種結果，在 for 跑完後還沒終止再輸出 Yes。但此種寫法要重複就會總是輸出 Yes，所以我改成檢查完所有頂點，都不是 No 後再輸出 Yes。(如圖 4 所示)

```

21 int main(){
22     //input
23     while(cin>>V>>E){
24         for(int i=0;i<E;i--){
31             //可能為非連通圖，故要遍訪所有頂點
32         }
33         for(int i=0;i<V;i--){
45             for(int i=0;i<V;i--){ G[i].clear() ;
46                 memset(color, 0, sizeof(color)) ;
47             }
48         }
49     }

```

圖 4 更新重複結構後的 main 函式

五、 顏色自選

在前面有提到，一個好的程式必須要有兼容性，必須要能夠在面對各種情況下還是

能完夠成執行並求解。正如前面所提到的可能出現不連通圖，所以必須遍歷頂點一樣；如果只限制在只能求是否為二分圖的話，豈不是太單一化了嗎？因此我決定再拓展這個程式，改成「能夠輸入要判斷的是幾分圖」。同樣地，會講述撰寫時遇上的一些問題。

1. 想法：

在輸入完圖的連接頂點後，再詢問需要嘗試以 C 種顏色填滿圖。然後需要一個以 C 種顏色做循環的迴圈，在需要填色時就使 `cur_color` 輪轉到下個顏色，並以 `cur_color` 對該頂點填色。同理，走訪完所有頂點，確認都無相異顏色過後則輸出 "The graph can be filled with C kinds of color." 反之則輸出 "cannot"。

2. 問題一：

顏色的循環必須從索引為「1」開始，因為初始化陣列時，我是將內部清空為零以完成初始化。假若顏色跳轉是從索引「0」開始，填入顏色 0 的頂點即等同於沒有填，所以必須修正為從 1 開始輪轉的相異顏色。

同時要注意的是，輪轉的顏色為 DFS 狀態遷移的前一個節點的下個顏色，而非一直往下個顏色輪轉。為此我苦惱了一會，後來才發現在下圖第 18 行是 `color[v]` 而非 `cur_color`。(如圖 5)

```

16  if(color[G[v][i]] == 0){
17      // 如果鄰接點尚未填色就用下個顏色來填
18      cur_color = (color[v]+1)%C ;
19      if(cur_color==0) cur_color=cur_color+C ;
20
21      if(!dfs(G[v][i])) return false ;
22  }
```

圖 5 顏色輪轉的部分程式

3. 問題二：

我在進行顏色是否相異的判斷時，誤將下圖的反白處寫為 `cur_color`。這在最一開始填顏色時不會出錯，因為顏色必定一直輪轉，故此頂點必定與相鄰頂點的顏色相異。但在最後(DFS 大多結束之後)收尾時，卻會導致新上色的頂點填完色後，把當前輪轉顏色視為相鄰的頂點，導致程式輸出錯誤。

為了除去這個 bug 花了些許心思，在查看填色結果是正確的後，發現原來是這裡的判斷出了問題，一切真相才水落石出。以後要更加細心地的注意這種細節，不要被看似正確的答案給騙了。(如圖 6 所示)

```

12  for(int i=0;i<G[v].size();i--){
13      // 如果鄰接的點顏色相同就傳回false
14      if(color[G[v][i]] == color[v] ) return false ;
15
16      if(color[G[v][i]] == 0){
23  }
```

圖 6 判斷顏色是否相異的部分程式

六、 小結 & 總述

到這裡這個程式需要應對的情況應該都涵蓋到了，特別注意是否為連通圖這部分，也自己加上了重複結構以及顏色自選的功能。我認為「有向圖」較少遇上，以及沒有必要刻意輸出各顏色的點(知道該圖為幾分圖後，填色就很簡單了)，所以就沒再實作出這兩部分。(不然輸入有向圖超麻煩的阿！)

歸納出一個很直觀的小性質，就是當你確定該圖為 k -分圖後，此圖亦可被所有大於 k 的自然數 n 個顏色給相異填滿。例如：看到本頁之第二筆測資的圖，在確認該圖為二分圖後，即得知其也能以三種、四種、五種等相異色填滿該圖。

另外，圖還有在此題相對不重要的幾個屬性沒提到：迴圈、多重邊、以及權重。如果有自己連到自己的自環的話，那該 k -分圖必不存在。如果存在多重邊，其在此題的意義等價於只有一條邊；各邊有權重的話，也不會影響到結果，因為這裡僅討論頂點間的連通與否而已。

伍、實作結果

將以表 1 示範幾筆測資：

<p>圖示 #1</p>	<p>輸入 #1</p> <p>Vertices : 6</p> <p>Edges : 6</p> <pre>0 2 0 5 1 4 1 5 2 3 2 4</pre> <p>Fill with 2 colors.</p>	<p>圖示 #2</p>	<p>輸入 #2</p> <p>Vertices : 8</p> <p>Edges : 6</p> <pre>1 5 2 4 6 4 4 3 3 7 7 2</pre> <p>Fill with 2 colors.</p>
<p>輸出 #1</p> <p>The graph cannot be filled with 2 kinds of color.</p>	<p>輸出 #2</p> <p>The graph can be filled with 2 kinds of color.</p>		
<p>圖示 #3</p>	<p>輸入 #3</p> <p>Vertices : 8</p> <p>Edges : 12</p> <pre>0 1 1 2 3 2 0 3 5 4 7 6 7 4 6 5 2 6 4 0 5 1 3 7</pre> <p>Fill with 2 colors.</p>	<p>圖示 #4</p>	<p>輸入 #4</p> <p>Vertices : 7</p> <p>Edges : 9</p> <pre>0 1 2 1 1 3 3 4 2 4 4 5 6 5 0 2 2 5</pre> <p>Fill with 3 colors.</p>
<p>輸出 #3</p> <p>The graph can be filled with 2 kinds of color.</p>	<p>輸出 #4 (錯誤)</p> <p>The graph cannot be filled with 3 kinds of color.</p>		

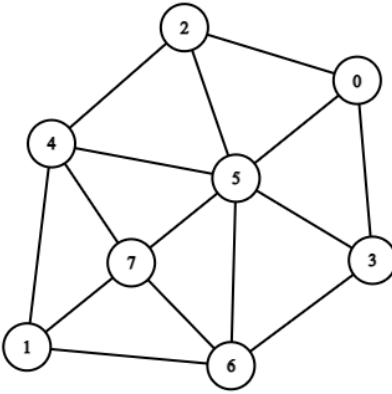
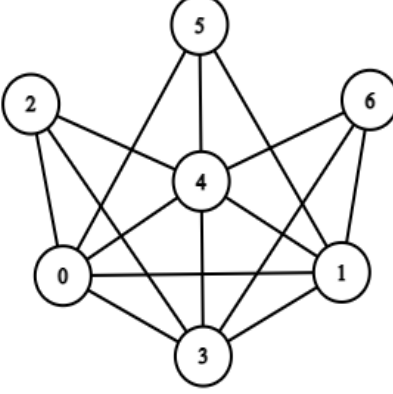
<p>圖示 #5</p> 	<p>輸入 #5</p> <p>Vertices : 8</p> <p>Edges : 15</p> <pre> 6 1 5 2 0 3 3 6 5 4 7 1 6 5 4 1 0 2 7 6 2 4 0 5 4 7 5 7 5 3 </pre> <p>Fill with 3 colors.</p>	<p>圖示 #6</p> 	<p>輸入 #6</p> <p>Vertices : 8</p> <p>Edges : 15</p> <pre> 2 0 5 1 6 1 9 2 3 2 4 1 6 3 3 4 4 2 6 4 3 1 3 0 0 1 5 4 4 0 </pre> <p>Fill with 3 colors.</p>
<p>輸出 #5 (錯誤)</p> <p>The graph cannot be filled with 3 kinds of color.</p>		<p>輸出 #6 (碰巧正確)</p> <p>The graph can be filled with 3 kinds of color.</p>	

表 1 自訂測資的輸出輸入以及圖例的表

(圖示製作：Graph Editor → https://csacademy.com/app/graph_editor/)

從上面的六筆測資中可以看出，有一半的輸出都不是正確的。上面那一串程式看似都做的完善了，但在輸入為三分圖時，卻沒有辦法得到正確的解答。我當然不會就此放棄，二分圖可以，三分圖卻不行，我馬上想到的是顏色輪轉又出了問題。

不出所料，正是顏色輪轉的時機不正確。以上頁的測資#4 為例，在節點 0 填入顏色 1 || 節點 1 填入顏色 2 || 節點 2 填入顏色 3 || 節點 4 填入顏色 1 || 接著要在節點 3 填入顏色 2 的時候，因為我程式的顏色輪轉只有在填新頂點時才會轉到下個顏色。這就導致節點 3 與

節點 1 都填入了顏色 2，使得在原本能以三個相異色填滿的測資#4 被誤判成不能填滿。

於是乎，我必須要再把我的程式維護的更好。我想出的解法如下：先將 dfs 到的空頂點填入當前的顏色(cur_color)，然後遍歷與其相鄰的所有頂點。若出現與其相同的顏色時，則輪轉到下一個顏色。等到每個顏色都輪完後(重複 C 次)，還是無法找到與相鄰頂點相異的顏色時，就 return false。(如圖 6)

```

9  bool dfs(int v){
10     color[v] = cur_color ;    // 以 cur_color 填滿頂點 v
11     int i=0, temp=C ;
12
13     while(i<G[v].size()){
14         // 如果鄰接的點顏色相同就輪轉到下個顏色
15         while(color[G[v][i]]==color[v] && temp--){
16             cur_color = (color[v]+1)%C ;
17             if(cur_color==0) cur_color=cur_color+C ;
18             color[v] = cur_color ;
19
20             // C 種顏色輪完即回傳false
21             if(temp==0)
22                 return false ;
23         }
24     }

```

圖 6 更新後的顏色輪轉程式

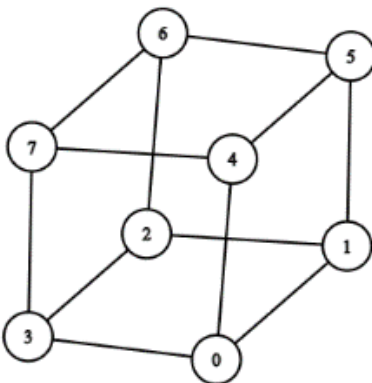
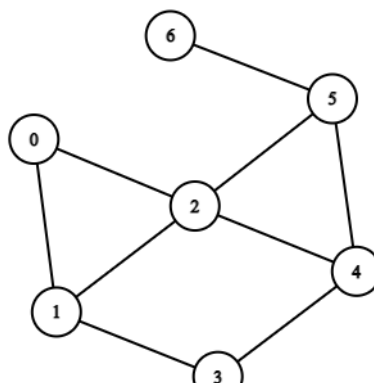
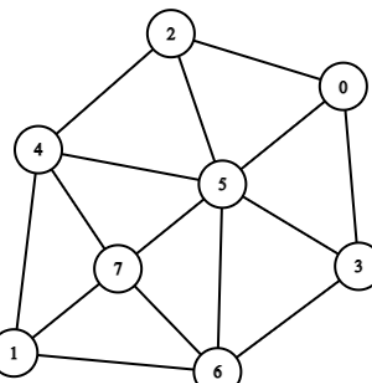
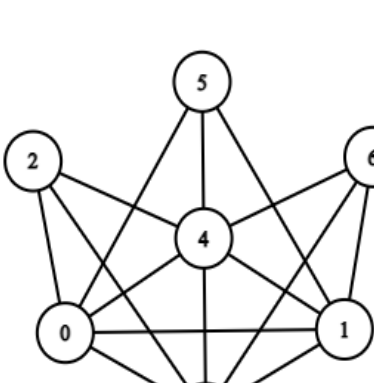
```

25  if(color[G[v][i]] == 0){
26      // 如果該接點尚未填色就用下個顏色來填
27      cur_color = (color[v]+1)%C ;
28      if(cur_color==0) cur_color=cur_color+C ;
29
30      if(!dfs(G[v][i])) return false ;
31  }
32  i++ ; //往下一個連接的頂點(索引值+1)
33  }
34  // 所有頂點都填好後即傳回 true
35  return true ;
36  }

```

圖 6 更新後的顏色輪轉程式 (續)

重新試測資：(如表二)

<div>圖示 #3</div> 	<div>輸入 #3</div> <div>Vertices : 8</div> <div>Edges : 12</div> <div>0 1 1 2</div> <div>3 2 0 3</div> <div>5 4 7 6</div> <div>7 4 6 5</div> <div>2 6 4 0</div> <div>5 1 3 7</div> <div>Fill with 2 colors.</div>	<div>圖示 #4</div> 	<div>輸入 #4</div> <div>Vertices : 7</div> <div>Edges : 9</div> <div>0 1 2 1</div> <div>1 3 3 4</div> <div>2 4 4 5</div> <div>6 5 0 2</div> <div>2 5</div> <div>Fill with 3 colors.</div>
<div>輸出 #3</div> <div>The graph can be filled with 2 kinds of color.</div>		<div>輸出 #4</div> <div>The graph can be filled with 3 kinds of color.</div>	
<div>圖示 #5</div> 	<div>輸入 #5</div> <div>Vertices : 8</div> <div>Edges : 15</div> <div>6 1 5 2</div> <div>0 3 3 6</div> <div>5 4 7 1</div> <div>6 5 4 1</div> <div>0 2 7 6</div> <div>2 4 0 5</div> <div>4 7 5 7</div> <div>5 3</div> <div>Fill with 3 colors.</div>	<div>圖示 #6</div> 	<div>輸入 #6</div> <div>Vertices : 8</div> <div>Edges : 15</div> <div>2 0 5 1</div> <div>6 1 9 2</div> <div>3 2 4 1</div> <div>6 3 3 4</div> <div>4 2 6 4</div> <div>3 1 3 0</div> <div>0 1 5 4</div> <div>4 0</div> <div>Fill with 3 colors.</div>
<div>輸出 #5</div> <div>The graph can be filled with 3 kinds of color.</div>		<div>輸出 #6</div> <div>The graph cannot be filled with 3 kinds of color.</div>	

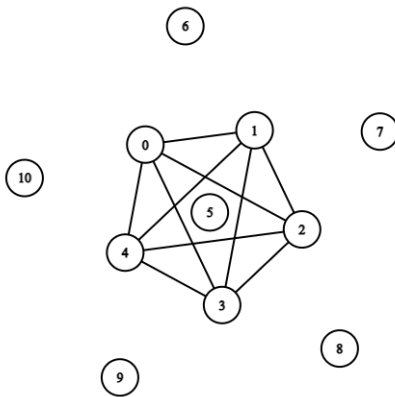
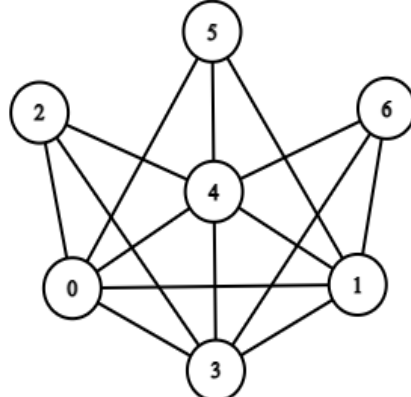
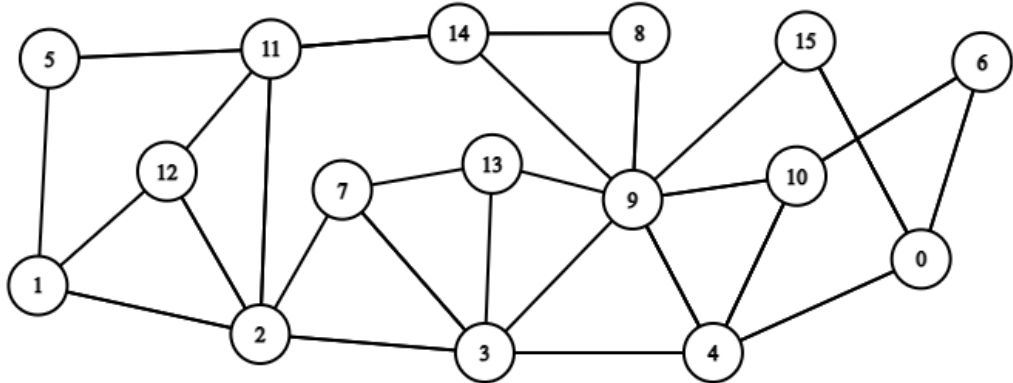
<div>圖示 #7</div> 	<div>輸入 #7</div> <div>Vertices : 11</div> <div>Edges : 10</div> <div>0 1 1 2</div> <div>2 3 3 4</div> <div>4 0 0 2</div> <div>2 4 4 1</div> <div>1 3 3 0</div> <div>Fill with 4 colors.</div>	<div>圖示 #8</div> 	<div>輸入 #8</div> <div>Vertices : 7</div> <div>Edges : 15</div> <div>2 0 5 1</div> <div>6 1 9 2</div> <div>3 2 4 1</div> <div>6 3 3 4</div> <div>4 2 6 4</div> <div>3 1 3 0</div> <div>0 1 5 4</div> <div>4 0</div> <div>Fill with 4 colors.</div>
<div>輸出 #7</div> <div>The graph cannot be filled with 4 kinds of color.</div>		<div>輸出 #8</div> <div>The graph can be filled with 4 kinds of color.</div>	
<div>圖示 #9</div> 		<div>輸入 #9</div> <div>Vertices : 16</div> <div>Edges : 27</div> <div>0 4 0 6</div> <div>0 15 1 2</div> <div>1 5 1 12</div> <div>2 3 2 7</div> <div>2 11 2 12</div> <div>3 4 3 7</div> <div>3 9 3 13</div> <div>4 9 4 10</div> <div>5 11 6 10</div> <div>7 13 8 9</div> <div>8 14 9 10</div> <div>9 13 9 14</div> <div>9 15 11 12</div> <div>11 14</div> <div>Fill with 3 colors.</div>	
<div>輸出 #9</div> <div>The graph can be filled with 3 kinds of color.</div>			

表 2 更新後自訂測資的輸出輸入以及圖例的表格

在經歷了反覆的修改，以及重重測資的磨練後，這支程式總算是趨於完善了。

底下我有放上一整支程式的面貌(如圖 7)，另外再附上 Github 連結及 Qrcode(如圖 8)。

Github 的連結：<https://reurl.cc/v5lg41>

```

1  #include<bits/stdc++.h>
2  using namespace std ;
3
4  vector<int> G[1000] ; // 圖
5  int V, E, C ; // V 頂點數, E 邊數, C 顏色數
6  int cur_color ; // 當前顏色
7  int color[1000] ; // 頂點 i 的顏色(1 or -1)
8
9  bool dfs(int v){
10 // 以 cur_color 填滿頂點 v
11 color[v] = cur_color ;
12 int i=0, temp=C ;
13
14 while(i<G[v].size()){
15 // 如果鄰接的點顏色相同就輪轉到下個顏色
16 while(color[G[v][i]]==color[v] && temp--){
17 cur_color = (color[v]+1)%C ;
18 if(cur_color==0) cur_color=cur_color+C ;
19 color[v] = cur_color ;
20
21 // C 種顏色輪完即回傳false
22 if(temp==0)
23 return false ;
24 }
25
26 if(color[G[v][i]] == 0){
27 // 如果鄰接點尚未填色就用下個顏色來填
28 cur_color = (color[v]+1)%C ;
29 if(cur_color==0) cur_color=cur_color+C ;
30
31 if(!dfs(G[v][i])) return false ;
32 }
33 i++ ; // 往下一個連接的頂點(索引值+1)
34 }
35 // 所有頂點都填好後即傳回 true
36 return true ;
37 }
38
39 int main (){
40 // input
41 cout << "Input the number of vertex in your graph : " ;
42 while(cin>>V){
43 cout << "Input the number of edge in your graph : " ;
44 cin >> E ;
45 cout << "Input the edges below : " << endl ;
46
47 for(int i=0;i<E;i++){
48 // 加上 s 到 t 的邊
49 int s, t ;
50 cin >> s >> t ;
51 G[s].push_back(t) ;
52 G[t].push_back(s) ;
53 // 因為是無向圖，再加上 t 到 s 的邊
54 }
55
56 cout << "Filled with _____ kinds of colors : " ;
57 cin >> C ;
58 cur_color = 1 ;
59
60 // 輸出所有頂點所連接到的各個頂點
61 for(int j=0;j<V;j++){
62 cout << "vertex " << j << " connect : " ;
63 for(int i=0;i<G[j].size();i++){
64 cout << G[j][i] << " " ;
65 }
66 cout << endl ;
67 }
68
69 // 可能為非連通圖，故要走訪所有頂點
70 for(int i=0;i<V;i++){
71 if(color[i]==0){
72 // 如果頂點尚未填色的就用 1 來填
73 if(!dfs(i)){
74 cout << "The graph cannot be filled with " ;
75 cout << C << " kinds of color." << endl ;
76 break ;
77 }
78 }
79 if(i==V-1){
80 cout << "The graph can be filled with " ;
81 cout << C << " kinds of color." << endl ;
82 break ;
83 }
84 }
85
86 // 串列跟對應顏色的初始化
87 for(int i=0;i<V;i++){
88 G[i].clear() ;
89 }
90 memset(color, 0, sizeof(color)) ;
91 cout << "---" << endl ;
92 cout << "Input the number of vertex in your graph : " ;
93 }
94
95

```



圖 8 Github 的 Qrcode 連結

圖 7 完整的程式樣貌

陸、討論總述

在此寫下對程式的想法及注意的點：

0. 建立 DFS 函式及各項變數宣告：

使用鄰接串列來儲存圖因為其相對於矩陣來的好 DFS，`cur_color` 為填入當前節點的顏色，若與其相鄰的頂點已經有該顏色了，則輪轉到下一個顏色。注意顏色須從 1 開始輪到詢問的總顏色數，否則填入 0 將被視為沒有填。

1. 輸入變數：

由於有向圖較少遇上，而無向邊的 $A \rightarrow B$ 等價於 $B \rightarrow A$ ，為了節省輸入的時間，選擇輸入的圖都必為無向圖。順帶一提，節點會從零開始到輸入的節點數，串列(節點)數量我設定為 1000 以內，不過在環境允許的情形下到 10 萬理應是沒有問題的。

2. DFS：

先以顏色 1 填入節點 0 後，就開始慢慢轉移下去。填完一個新節點後，馬上搜尋與其相鄰的頂點，有相同的就輪轉到下個顏色。開始輪轉的顏色應為上個填好的節點之顏色，找到相異色後即填入並呼叫下個節點做 DFS，否則 `return false` 來中止並輸出結果。

3. 輸出及初始化：

依照最後都填完所有頂點後 DFS 函式回傳的布林值來判斷是否可被 C 種顏色填滿。並且使記錄圖的鄰接串列清空、節點填色的矩陣歸零。

柒、自身看法

正好課堂上正在學習排列組合，並且程式也學完圖論了，就心血來潮實作了這個程式。雖然在 debug 顏色輪轉的地方時一再受挫，不過我還是耐心地找出 bug 並把程式修正成完善的版本了。這種感覺很像接案子，發現了生活中一些小地方、小問題，並且用程式準確的輔助我們完成工作。這比一直刷 online judge 的題目有趣多了，更有寫出來的成就感。我會再更細心的留意周遭，並且用程式這個方法改善那些問題。

捌、資料來源

【圖示製作】Graph Editor→https://csacademy.com/app/graph_editor/

【圖 1 DFS 狀態遷移的順序】維基百科→https://en.wikipedia.org/wiki/Depth-first_search

【概念及程式碼參考】培養與鍛鍊程式設計的邏輯腦 秋葉拓哉 著 博碩文化 譯