110 學年度第二學期

圖形理論 Project 報告書

學號：N26114976

姓名：王梓帆

111 年 03 月 17 日

目 錄

內容

表目錄

圖目錄

第一章 前言

1-1　專案動機與目的

電腦的運算速度愈來愈快，也讓很多原本以圖形為基礎的演算法，得以在各式各樣的軟體語言上實踐，圖形理論的演算法也被用來解很多最佳化問題，在未來，更有可能在量子領域被廣泛運用，圖（Graph）是圖形理論的主要研究對象，由若干給定的頂點（vertex）及連接兩頂點的邊（edge）所構成的圖形，這種圖形通常用來描述某些事物之間的某種特定關係。頂點用於代表事物，連接兩頂點的邊，則用來表示兩個事物的關係。

課堂上教授也由提到，圖形理論起源於著名的柯尼斯堡七橋問題。該問題於 1736 年被歐拉解決，因此普遍認為歐拉是圖形理論的創始人。後人也繼續進行了多方的探討，才讓這個數學方法被廣泛使用，本次的專案雖然探討的是很基礎的問題，也希望在這個基礎上，能夠討論圖形理論中二部圖（Bipartite Graph）的特徵，以及不同實踐演算法對記憶體占比與時間的影響。

1-2　專案內容

本次專案的內容，是在 Microsoft Windows 的環境下，透過 Cygwin 模擬器，模擬類 UNIX 系統，並在該模擬器上以 C/C++ 程式，判斷輸入的圖形是否為二部圖（Bipartite Graph），並建立在這個基礎上，做各方面的延伸與探討。

第二章 演算法概述

2-1 二部圖（Bipartite Graph）的特徵及重要性

在圖形理論中，二部圖是一類特殊的圖，又稱為二分圖、偶圖、雙分圖。二部圖的頂點可以分成兩個互斥的獨立集合（Independent Set）U 和 V，使得所有邊都是連結一個 U 中的點和一個 V 中的點。頂點集 U、V 被稱為是圖的兩個部分。二部圖也可以等價被定義成，**圖中所有的環都有偶數頂點**，也就是教授在課堂上提供給我們的加分際會，需要去證明的等價條件。

以圖形的角度看，我們也可以將 U 和 V 當成著色問題來看： U 中所有頂點為藍色，V 中所有頂點為紅色；只要每條邊的兩個端點顏色不同，就符合圖著色問題的要求。相反的，非二部圖則無法用兩種顏色完成著色問題，例如 K_3 ，即內含 3 頂點的完全圖（Complete Graph），將其中一個頂點標示為藍色，並且另外一個標示為紅色後，第三個頂點與上述具有兩個顏色的頂點相連，無法再對它塗上藍色或紅色。

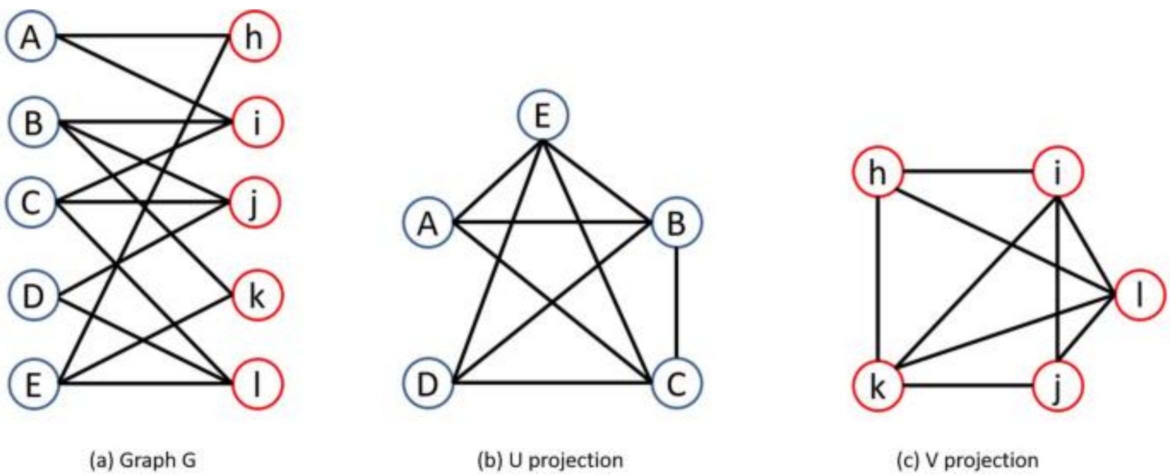


圖 2-1-1 二部圖範例

附圖便是一個二部圖的範例，我們可以看到它可以被分成兩個互斥的獨立集 U 和 V，使得所有邊都是連結一個 U 中的點和一個 V 中的點。我們也可以將它寫作

$$\mathbf{G} = (\mathbf{U}, \mathbf{V}, \mathbf{E}) \tag{1}$$

包含了獨立集 U 和 V，以及邊 E 的資訊，且二部圖 G，在 G 的一個子圖 M 中，M 的邊集中的任意兩條邊都沒有共同的端點，則稱 M 是一個匹配（Match）。

2-2　DFS（Depth-First Search）深度優先搜尋演算法

是一種用來閱覽一個圖（Graph）的演算法。由圖的某一點當成根（Root）來開始探尋，先探尋邊（Edge）上未搜尋的頂點（Vertex），並盡可能往深處搜索，直到該頂點的所有邊上頂點都搜索完成時，就回溯（Backtracking）到前一個頂點，重覆探尋未搜尋的節點，直到完成目的（以本專案來看，就是找到第三種著色）或遍尋全部頂點為止。

深度優先搜尋法屬於盲目搜索（uninformed search）是利用堆疊（Stack）來處理，通常以遞迴的方式呈現。

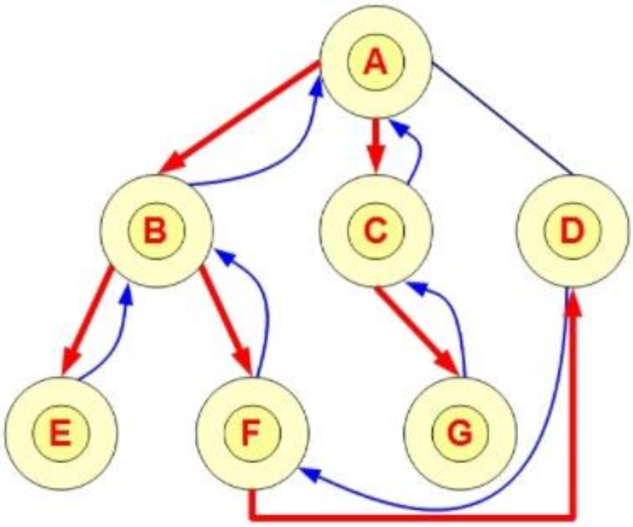


圖 2-2-1　DFS 演算法範例

以附圖為例，該圖會從 A→B→E，再由 B→F→D，並從 D→F→B→A 走回出發點，最後由 A 走到 G，再走回 A，完成全部的閱覽。簡單來說，深度優先搜尋法是以「**先深後廣**」的方式，去遍歷整個資料集，得到最終的辨識結果，也適合用來解決圖形是否為二部圖的圖形問題。

2-3　BFS（Breadth -First Search）深度優先搜尋演算法

廣度優先搜尋法，是一種圖形（graph）搜索演算法。從圖的某一節點（vertex）開始走訪，接著走訪此一節點所有相鄰且未拜訪過的節點，由走訪過的節點繼續進行搜尋。以圖形（graph）來說，就是把把同一階層（level）的頂點走完，再繼續向下一個階層搜尋，直到達到目的，或遍尋全部頂點。

廣度優先搜尋法屬於盲目搜索（uninformed search）是利用佇列（Queue）來處理，通常以迴圈的方式呈現。

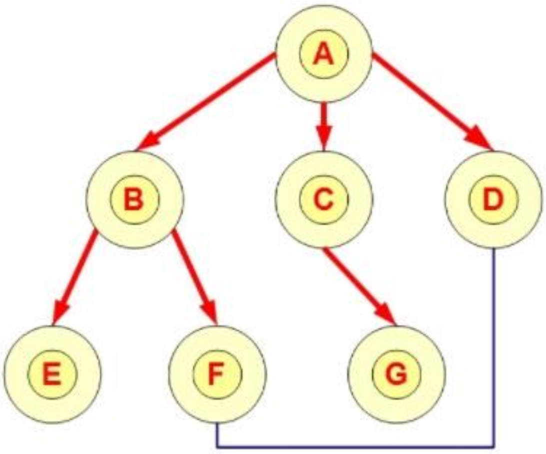


圖 2-3-1　BFS 演算法範例

以附圖為例，該圖會從 A 同時走向 B、C 與 D，再由 B 展開至 E 與 F，最後則是由 C 至 G。簡單來說，廣度優先搜尋法是以「**先廣後深**」的方式，去遍歷整個資料集，得到最終的辨識結果，和深度優先搜尋法一樣，適合用來解決圖形是否為二部圖的圖形問題。

2-4　最大匹配（Maximum Matching）

一個圖的所有匹配中，邊數最多的匹配稱為這個圖的最大匹配。前面有提到一個圖是一個匹配（或稱獨立邊集）是指這個圖之中，任意兩條邊都沒有公共的頂點。這時每個頂點都至多連出一條邊，而每一條邊都將一對頂點相匹配。在二部圖很常延伸討論這個問題。

2-4-1　**二部圖中的匹配**

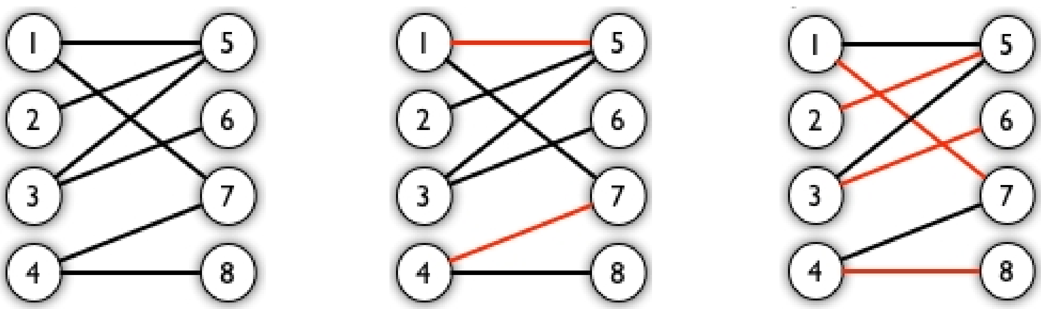


圖 2-4-1　二分圖匹配範例

給定一個二部圖 G，在 G 的一個子圖 M 中，M 的邊集{E}中的任意兩條邊都不依附於同一個頂點，則稱 M 是一個匹配，如圖 2-4-2，第二與第三張圖紅色的邊，皆為第一張圖的匹配。

2-4-2 **最大匹配**

一個圖所有匹配中，所含匹配邊數最多的匹配，稱為這個圖的最大匹配。以圖 2-4-3 為例，最右邊的突擊為最左邊的圖的最大匹配。

2-4-3 **最大流（Maximum Flow）問題**

一般來說 Flow Network 都會被規劃成一個有向圖，與常見的有向圖不同的是，Flow Network 多了容量（capacity）的限制。Flow Network 中會定義 source（s），用於產生 flow，以及用於接收 flow 的 sink（t）。Flow Network 的重點是找出最大可容納流量，可以從兩個方向來觀察：第一個是 source 一次最多可以放出多少流量，第二則是 t 可以一次最多能承接多大容量。

最大流（Maximum Flow）問題顧名思義，便是給定一個 Flow Network，要找到這個 Flow Network 最大導入流量的問題，我們可以計算 source 端總共產生了多少流量，或是 sink 端接收了多少流量，而兩者必定相同，即為這張圖的最大流量。

送出流量必定符合下面兩項規定：第一，容量條件（Capacity Conditions），送進管線的流量不能超過管線的容量，且最後的結果不能為負；第二，守恆條件（Conservation Conditions），從圖中任一點的流入一定要等於流出。

在上述限制之下，我們可以求得 Maximum Flow 的上限，只要有流從 source 端到 sink 端，那一定會經過圖的某個橫切面，因此可以選擇圖中某個橫面將圖切成兩半，讓 source 端與 sink 端分別在兩邊。且被切過的邊都有方向性，裡面的流動的水流大小只能灌滿管線但不能超過它，此時截面便形成了上限。由於截面可以任意形成，因此每個截面都會提供一個上限的數值，只要從這些數值中找到最小的數字，就是上限的極值，如果能找到最小的上限，就會是我們要求的流量最大值。

2-4-4 **Ford-Fulkerson 演算法**

Ford-Fulkerson 演算法是用來解最大流(Maximum Flow)問題相當著名的演算法，先順著方向將流量灌入，再利用退回的機制找出其他導流的可能性。

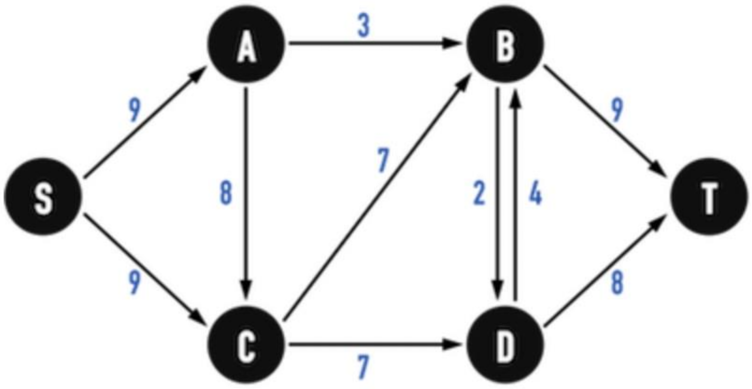


圖 2-4-4　Flow Network 範例

在演算法執行的過程中，會使用到 Residue 的概念，它的功用是將每條線路的殘餘流量記錄下來，從而發展出殘餘圖（Residual Graph）。這個演算法就是反覆在兩個圖上面做操作，從而求出最佳解。

Residual Graph 的記錄方法如下：

- 原本圖上的 source、sink 及點的位置不變，但是 edge 需要做一點調整。
- 若是管線中的流量還沒有滿，就要記錄下來。
- 為管線提供 undo flow 的機制，同時也要記錄可以退回的流量單位。

Residual Graph 完成之後，接下來就要檢視是否有沒有辦法在上面繼續增加流量。此時我們要找的，就是 Residual Graph 中，從 source 到 sink 的路徑，找到路徑並把流量灌入之後，就更新原來的 Flow Network ，更新後的 Flow Network 又可以再產生新的 Residual Graph，如此反覆執行，就能找到圖的最大流量。找到路徑將流量灌入管線的動作稱為 Augmenting Path。

當 Ford-Fulkerson algorithm 找不到可以從 s 到 t 的路徑時，演算法停止，此時 Residual Graph 中 s 只能走到圖上的某一個點，此時圖就自然形成一個 cut，分割線經過的路徑流量算出來的上限會與 flow value 相等，這就是最大流最小割（maximum flow minimum cut）。

2-4-5　以最大流問題的角度解 Bipartite 最大匹配問題

這個問題最困難的是畫出相應的 Flow Network，只要能畫出 Flow Network 就能用上述方法解決問題。解題思維如下，先仿照 Flow Network 為原本的圖增加 source 和 sink，並強制給予 edge 方向性。source 和 sink 分別連接二部圖中不同集合的點，並給予 edge capacity 設為 1。再透過 Ford-Fulkerson 演算法，即可求得對於 Source 和 Sink 的 Maximum flow 即為該二部圖的最大匹配。

2-5　無向圖中的相連部件（Connected Components）

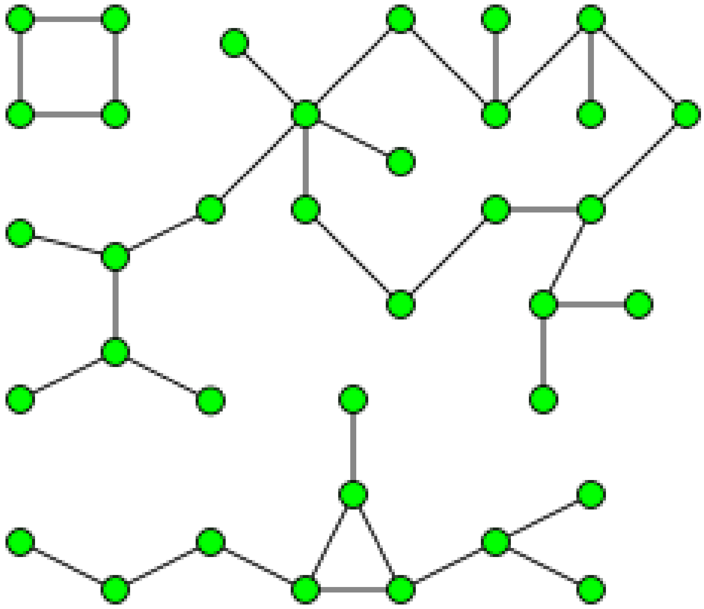


圖 2-5-1 相連部件（Connected Components）範例

教授於課堂中，也討論過圖中相連部件數量的問題，所謂的相連，就表示兩點之間存在 path，將兩點相連，而不再與其他點相連的子圖，我們就叫他部件(Component)，我們也很常探討無相圖中的分群問題，在判斷有幾個相連部件的部分，我所使用的是 Union Find 演算法，他一開始是用來討論不共點的兩子集，在圖論中，可以用來探討不相連子圖的數量，也就可以找出 Components 的數量，與該 Component 之中有哪些 vertex。

2-5-1　**Union Find 演算法**

Union Find 演算法用來處理不共點集合的資料，而 Component 正好就是這樣的資料。該演算法主要做兩件事情，首先是 **Find** 用來判斷點屬於哪個子集，對應到圖論的問題，就變成是在判斷 vertex 屬於哪個子圖，也可以來判斷兩個點是否屬於同一個子集或子圖。再來則是 **Union**，如果確認兩個子圖屬於同一個子圖，我們可以將兩個子圖合併。最後透過這個演算法，我們可以判斷哪些點相連，以及總共有幾組不相連的子圖，藉此判斷有多少 Components。

第三章 程式碼實現

3-1　輸入的介紹

輸入放在 graph_input.h 檔裡面，以圖 3-1-1 為例，輸入格式如下：

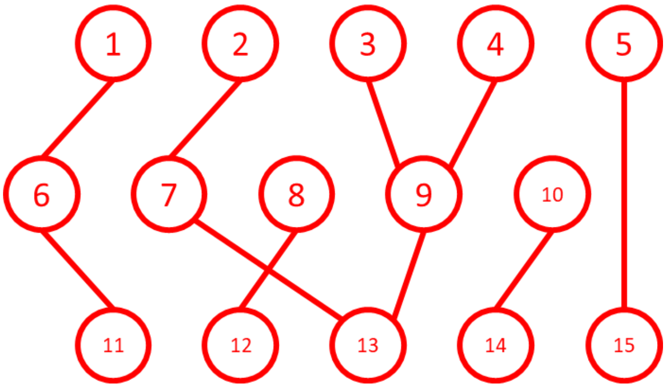


圖 3-1-1 助教給定的 Benchmark 1

```
vector<vector<bool>> G = { {0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0},
                           {0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0},
                           {0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0},
                           {0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0},
                           {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1},
                           {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0},
                           {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0},
                           {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0},
                           {0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0},
                           {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0},
                           {0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0},
                           {0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0},
                           {0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0},
                           {0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0},
                           {0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}};
```

G 會被設定為一個 V*V 的二維向量，V 為頂點的個數，V[A][B]，則表示欲判斷的圖第 A 個頂點與第 B 個頂點相連。

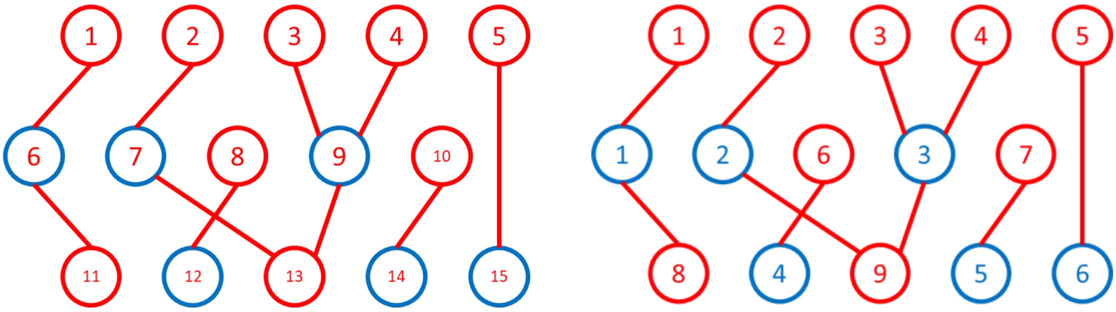


圖 3-1-2 Benchmark 為二部圖的情況

若 G 經過判斷為二部圖，將重新定義點的排序，由上圖左至上圖右，並定義一個 M*N 的二維陣列 bpGraph，M 為紅色頂點個數，N 為藍色頂點個數，若 bpGraph[M][N] 為 1，則表示 M, N 相鄰，兩者有 edge，否則無。

```
vector<vector<bool>> bpGraph = {{1, 0, 0, 0, 0, 0},
                                 {0, 1, 0, 0, 0, 0},
                                 {0, 0, 1, 0, 0, 0},
                                 {0, 0, 1, 0, 0, 0},
                                 {0, 0, 0, 0, 0, 1},
                                 {0, 0, 0, 1, 0, 0},
                                 {0, 0, 0, 0, 1, 0},
                                 {1, 0, 0, 0, 0, 0},
                                 {0, 1, 1, 0, 0, 0}};
```

最後則是所有 edge 的集合，將會存成一個 N*2 的二維向量，N 表示圖 G 的 edge 數，每一 edge 兩端的頂點將被存在這個陣列裡。(註：這邊的 0 到 19 是圖上標示的 1 到 20，是因為這樣陣列定義完比較容易用迴圈做，輸出時會全部加一回來。)

```
vector<vector<int>> edges = {{0, 5},
                             {1, 6},
                             {2, 8},
                             {3, 8},
                             {4, 14},
                             {5, 10},
                             {6, 12},
                             {7, 11},
                             {8, 12},
                             {9, 13}};
```

3-2 Solution Class

表 3-2-1 UML Chart

Solution
+: bool isBipartite_BFS(vector<vector<bool>>, int)
+: isBipartite_DFS(vector<vector<bool>>)
+: bool bpm(vector<vector<bool>>, int, bool, int)
+: int maxBPM(vector<vector<bool>>)
+: int merge(vector<int>, int)
+: int connectedcomponents(int, vector<vector<int> >&, vector<vector<bool> >)
+: static vector<int> V_color;

3-3 二部圖（Bipartite Graph）的判斷

二部圖的判斷由 Solution Class 來進行，分別透過深度優先搜尋與廣度優先搜尋，兩種方法來比較，看精準度與執行時間。

3-3-1 深度優先搜尋法（Depth-First Search）

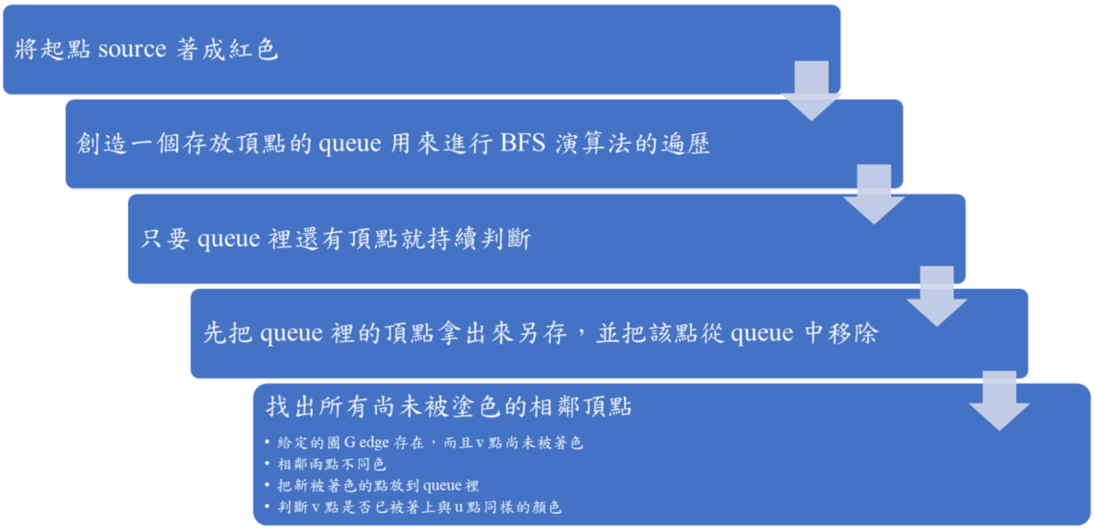
演算法實踐流程如下：

表 3-3-1 深度優先搜尋法流程

3-3-2　廣度優先搜尋法（Breadth-First Search）

演算法實踐流程如下：

表 3-3-2　廣度優先搜尋法流程



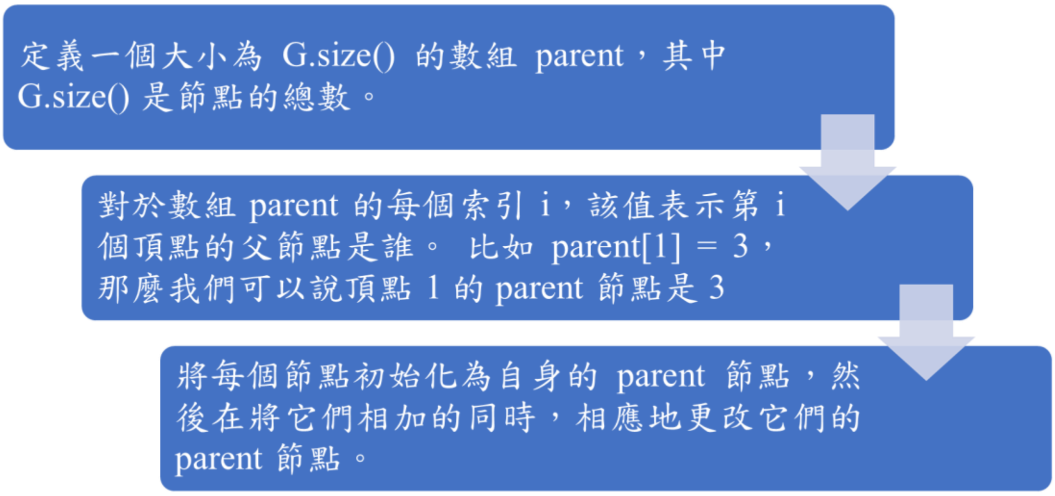
3-4　最大匹配數的計算

表 3-4-1　二部圖最大匹配：最大流演算法流程

Bipartite Match：如果和點 u 可能匹配（和點 u 相連），則 bpm 判斷為 True
• 將所有著藍色的點拿來做測試 • 如果紅色的點 u，和藍色的點 v 相連，且 v 尚未被其他點相連 　　• 先將 v 標示為已配對 　　• 如果藍色的點 v 沒有被其他點相連，或者先前與藍色的點相連的紅點（即 matchR[v]）有其他可以相連的點。 　　• 由於 v 在上一行中被標記為已配對，因此 matchR[v] 將不會再次遞迴點 v
Maximum Match：回傳最大匹配數
• 儲存與藍色的點相連的紅點 matchR[i] 是與藍色的點 i 相連的紅點數若為 -1 則該點沒有被相連 • 初始條件：所有的 v 點都可以被連線 　　• 對下一個紅點 u 初始化為尚未連接 　　• 判斷紅點 u 是否匹配，有的話匹配數加一。

3-5 相連部件數的計算

表 3-5-1 不共點子集演算法流程



4-1 輸出結果

4-1-1 輸出結果介紹

```
$ ./GT_Project_1.exe
Which graph would you like to input:
1
Result of the test
=============================
According to BFS Algorithm,
The graph given is BIPARTITE.
Time Consumed by BFS: 0.0373ms
-----------------------------
According to DFS Algorithm,
The graph given is BIPARTITE.
Time Consumed by DFS: 0.2529ms
-----------------------------
Red Group Members: 1,2,3,4,5,8,10,11,13
Blue Group Members: 6,7,9,12,14,15
-----------------------------
Maximum Matching is 6
-----------------------------
Components => 1 6 11
Components => 8 12
Components => 2 3 4 7 9 13
Components => 10 14
Components => 5 15
-----------------------------
There are 5 components.
=============================
```

圖 4-1-1 輸出結果樣式

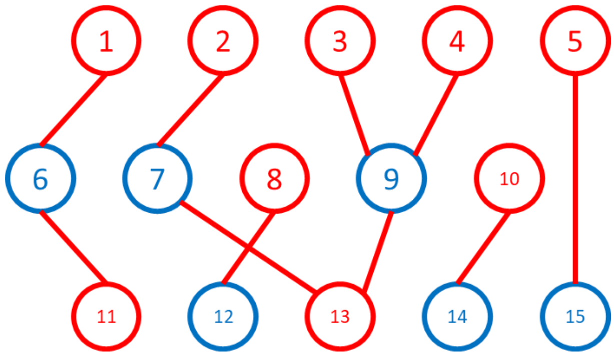


圖 4-1-2 二部圖著色問題

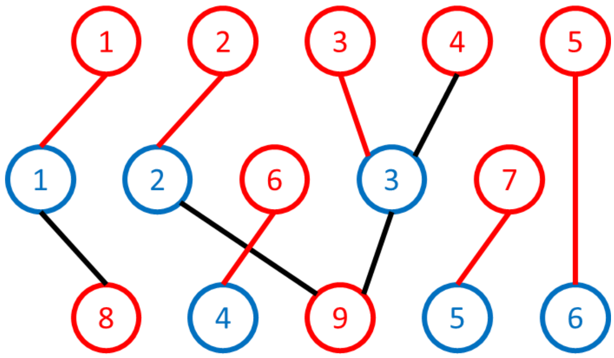


圖 4-1-3 二部圖最大匹配（黑色無匹配）

輸出會呈現的有：

1. 以 BFS 判斷輸入是否為二部圖並附上時間紀錄。
2. 以 BFS 判斷輸入是否為二部圖並附上時間紀錄。
3. 二部圖中，兩子圖內的頂點。
4. 最大匹配數。
5. 輸入圖所含部件數。
6. 各部件內的頂點。

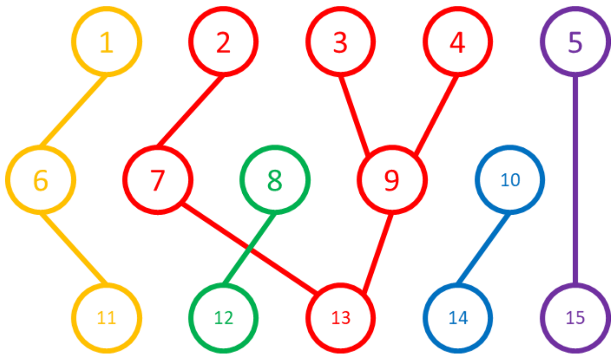
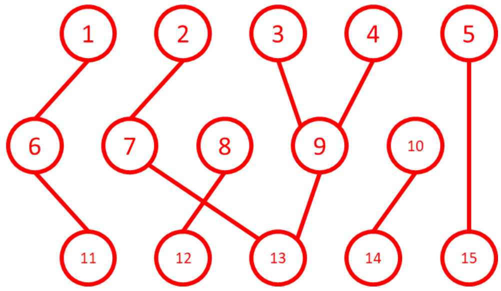


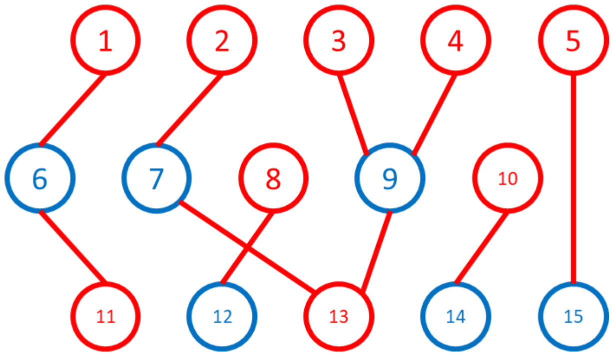
圖 4-1-4 輸入圖部件數量

4-1-2 所有輸出結果

表 4-1-1 Benchmark 1

INPUT

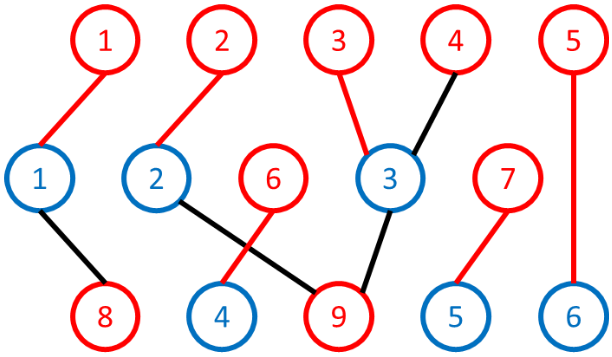




IS BIPARTITE!

Shortest Time Consume: 37.2 us

```
$ ./GT_Project_1.exe
Which graph would you like to input:
1
Result of the test
==============================
According to BFS Algorithm,
The graph given is BIPARTITE.
Time Consumed by BFS: 0.0373ms
------------------------------
According to DFS Algorithm,
The graph given is BIPARTITE.
Time Consumed by DFS: 0.2529ms
------------------------------
Red Group Members: 1,2,3,4,5,8,10,11,13
Blue Group Members: 6,7,9,12,14,15
------------------------------
Maximum Matching is 6
------------------------------
Components => 1 6 11
Components => 8 12
Components => 2 3 4 7 9 13
Components => 10 14
Components => 5 15
------------------------------
There are 5 components.
==============================
```



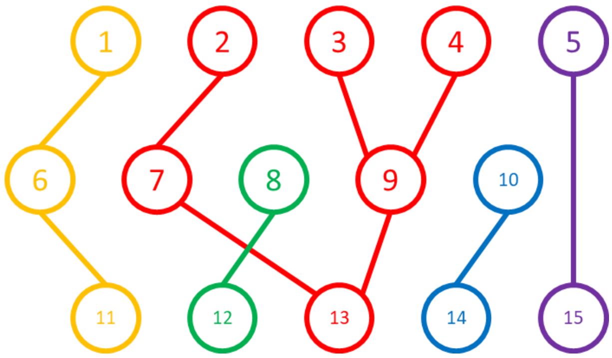
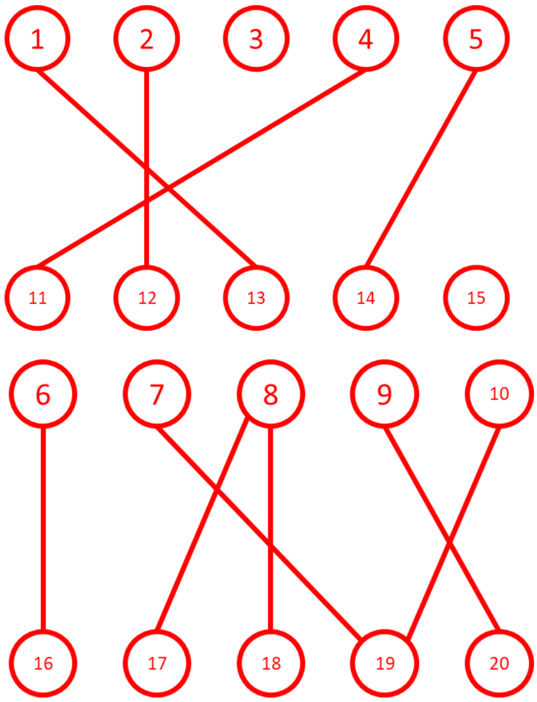
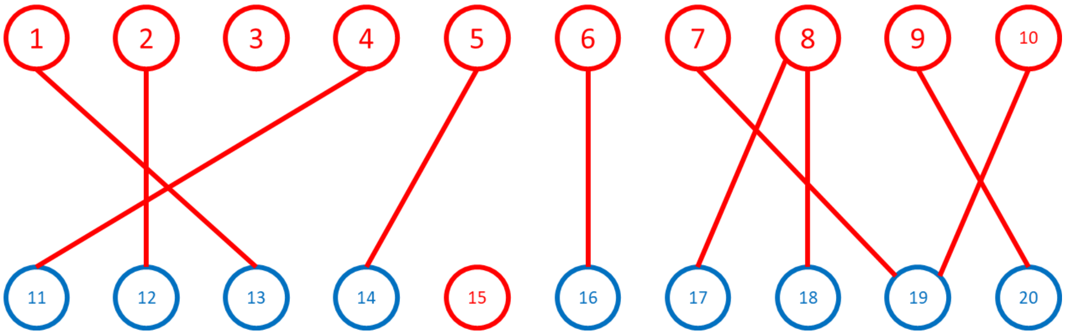
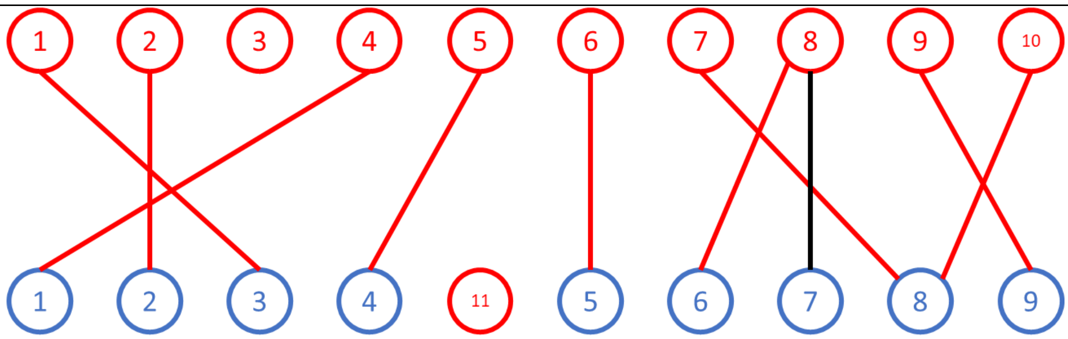


表 4-1-2　Benchmark 2

```
$ ./GT_Project_1.exe
Which graph would you like to input:
2
Result of the test
============================
According to BFS Algorithm,
The graph given is BIPARTITE.
Time Consumed by BFS: 0.0512ms
----------------------------
According to DFS Algorithm,
The graph given is BIPARTITE.
Time Consumed by DFS: 0.3294ms
----------------------------
Red Group Members: 1,2,3,4,5,6,7,8,9,10,15
Blue Group Members: 11,12,13,14,16,17,18,19,20
----------------------------
Maximum Matching is 8
----------------------------
Components => 3
Components => 4 11
Components => 2 12
Components => 1 13
Components => 5 14
Components => 15
Components => 6 16
Components => 8 17 18
Components => 7 10 19
Components => 9 20
----------------------------
There are 10 components.
============================
```







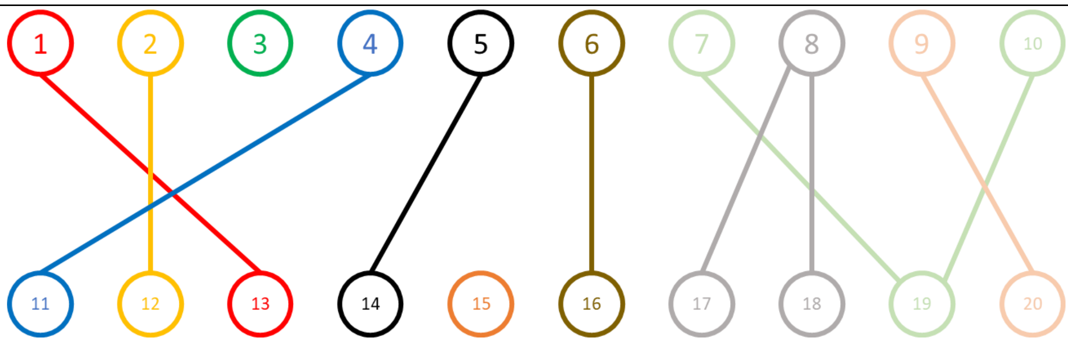
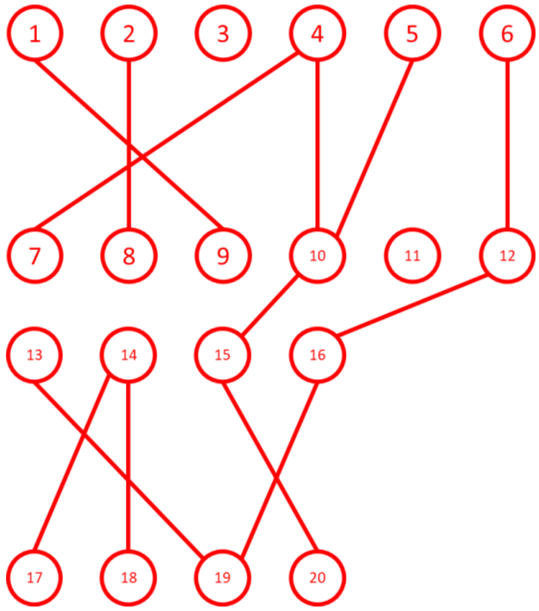
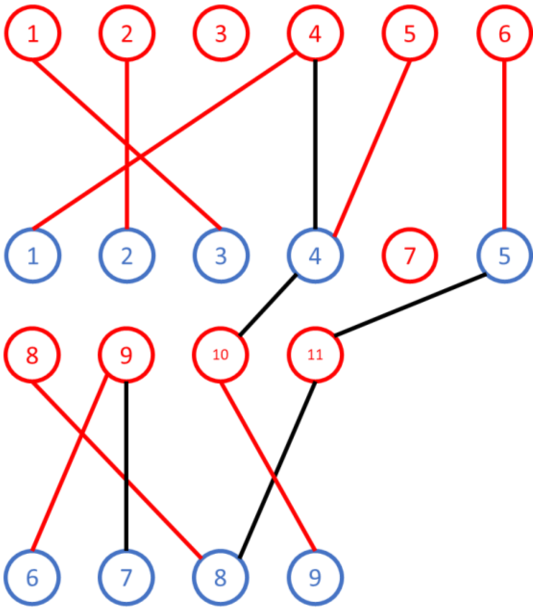


表 4-1-3 Benchmark 3

INPUT





IS BIPARTITE!

Shortest Time Consume: 49.6 us

```
Which graph would you like to input:
3
Result of the test
==============================
According to BFS Algorithm,
The graph given is BIPARTITE.
Time Consumed by BFS: 0.0496ms
------------------------------
According to DFS Algorithm,
The graph given is BIPARTITE.
Time Consumed by DFS: 0.2856ms
------------------------------
Red Group Members: 1,2,3,4,5,6,11,13,14,15,16
Blue Group Members: 7,8,9,10,12,17,18,19,20
------------------------------
Maximum Matching is 8
------------------------------
Components => 3
Components => 2 8
Components => 1 9
Components => 11
Components => 14 17 18
Components => 6 12 13 16 19
Components => 4 5 7 10 15 20
------------------------------
There are 7 components.
==============================
```

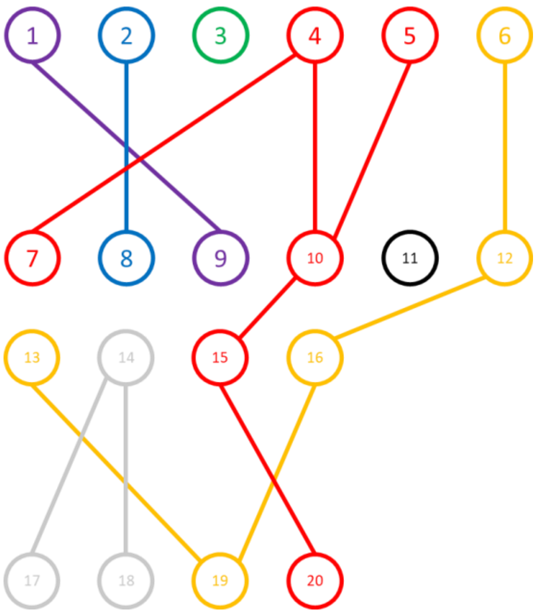


表 4-1-4 Benchmark 4 to 10

Benchmark 4

```
Which graph would you like to input:
4
Result of the test
============================
According to BFS Algorithm,
The graph given is BIPARTITE.
Time Consumed by BFS: 0.0479ms
----------------------------
According to DFS Algorithm,
The graph given is BIPARTITE.
Time Consumed by DFS: 0.3712ms
----------------------------
Red Group Members: 1,4,6,7,11,12,14,16,17,19
Blue Group Members: 2,3,5,8,9,10,13,15,18,20
----------------------------
Maximum Matching is 10
----------------------------
Components => 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
----------------------------
There are 1 components.
============================
```

Benchmark 5

```
Which graph would you like to input:
5
Result of the test
============================
According to BFS Algorithm,
The graph given is NOT BIPARTITE.
Time Consumed by BFS: 0.0394ms
----------------------------
According to DFS Algorithm,
The graph given is NOT BIPARTITE.
Time Consumed by DFS: 0.1016ms
----------------------------
It cannot divide into 2 Groups!
----------------------------
It is not Bipartite!
Please enter bipartite graph to calculate Maximum Match!
----------------------------
Components => 1 2 3 4 5 11 12 13 14 15
Components => 6 16
Components => 8 17 18
Components => 7 10 19
Components => 9 20
----------------------------
There are 5 components.
============================
```

Benchmark 6

```
$ ./GT_Project_1.exe
Which graph would you like to input:
6
Result of the test
============================
According to BFS Algorithm,
The graph given is NOT BIPARTITE.
Time Consumed by BFS: 0.0307ms
----------------------------
According to DFS Algorithm,
The graph given is NOT BIPARTITE.
Time Consumed by DFS: 0.1331ms
----------------------------
It cannot divide into 2 Groups!
----------------------------
It is not Bipartite!
Please enter bipartite graph to calculate Maximum Match!
----------------------------
Components => 1 6 11
Components => 2 3 4 7 8 9 12 13
Components => 10 14
Components => 5 15
----------------------------
There are 4 components.
============================
```

Benchmark 7

```
Which graph would you like to input:
7
Result of the test
============================
According to BFS Algorithm,
The graph given is BIPARTITE.
Time Consumed by BFS: 0.0398ms
----------------------------
According to DFS Algorithm,
The graph given is BIPARTITE.
Time Consumed by DFS: 0.2195ms
----------------------------
Red Group Members: 1,2,3,4,5,10,11,12,13
Blue Group Members: 6,7,8,9,14,15
----------------------------
Maximum Matching is 6
----------------------------
Components => 1 2 3 4 5 6 7 8 9 11 12 13 15
Components => 10 14
----------------------------
There are 2 components.
============================
```

Benchmark 8

```
Which graph would you like to input:
8
Result of the test
============================
According to BFS Algorithm,
The graph given is BIPARTITE.
Time Consumed by BFS: 0.0492ms
----------------------------
According to DFS Algorithm,
The graph given is BIPARTITE.
Time Consumed by DFS: 0.3512ms
----------------------------
Red Group Members: 1,4,5,6,10,11,12,13,16,17
Blue Group Members: 2,3,7,8,9,14,15,18,19,20
----------------------------
Maximum Matching is 8
----------------------------
Components => 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
----------------------------
There are 1 components.
============================
```

Benchmark 9

```
Which graph would you like to input:
9
Result of the test
============================
According to BFS Algorithm,
The graph given is NOT BIPARTITE.
Time Consumed by BFS: 0.0365ms
----------------------------
According to DFS Algorithm,
The graph given is NOT BIPARTITE.
Time Consumed by DFS: 0.2703ms
----------------------------
It cannot divide into 2 Groups!
----------------------------
It is not Bipartite!
Please enter bipartite graph to calculate Maximum Match!
----------------------------
Components => 4 11
Components => 2 12
Components => 1 13
Components => 3 5 6 7 8 9 10 14 15 16 17 18 19 20
----------------------------
There are 4 components.
============================
```

Benchmark 10

```
Which graph would you like to input:
10
Result of the test
============================
According to BFS Algorithm,
The graph given is BIPARTITE.
Time Consumed by BFS: 0.0481ms
----------------------------
According to DFS Algorithm,
The graph given is BIPARTITE.
Time Consumed by DFS: 0.3111ms
----------------------------
Red Group Members: 1,2,3,4,5,9,16,17,18,19
Blue Group Members: 6,7,8,10,11,12,13,14,15,20
----------------------------
Maximum Matching is 10
----------------------------
Components => 4 11
Components => 2 12
Components => 1 13
Components => 3 5 6 7 8 9 10 14 15 16 17 18 19 20
----------------------------
There are 4 components.
============================
```

4-2 問題與檢討

總體來說，我們發現到在這次的 Project 中，廣度優先的演算法具有更快的執行速度，但這可能一跟我使用到的寫法有關，另外，在這次的 Project 中，因為是自己一個人一組的關係，學到了很多，包含 BONUS 的發想、判斷的精準度與時間，芬黨的撰寫與建構，都比起想像中的還要花時間。

除了執行時間，還會發現到這次作業的 Benchmark 中，有很多圖是相關聯的，像是圖 2、9 跟 10，明明都只差幾個 edge，卻有完全不同的判斷結果，也讓人體會到圖論這門科學理論，是非常深奧且需要花時間詳讀的，希望在之後的報告與 Project 中都能學到很多嶄新的知識。