

元 智 大 學

資 訊 工 程 學 系

專題製作成果報告

利用 Hungarian method 及 Edmonds-
Karp algorithm 實作 maximum
bipartite matching

專 題 生：賴仲倫 、張宇越

學 號：1083301、1081542

指導教授：張經略 教授

中 華 民 國 111 年 12 月

目錄

一、 前言	2
二、 文獻探討	3
三、 研究方法	5
3.1、 初始架構 - 輸入設定.....	5
3.2、 初始架構 - 初始值設定與宣告.....	5
3.3、 方法流程 - Hungarian Algorithm.....	6
3.4、 方法流程 - Edmonds Karp Algorithm.....	9
3.5、 測資來源.....	10
四、 討論與分析	12
五、 結論.....	16
六、 References	19

一、前言

在圖論中，bipartite graph 是特殊的圖，又稱為二分圖、偶圖、二部圖。在 bipartite graph 中的點會被分成 V_0 、 V_1 兩個互斥集，同個集合中的點不會有邊相連。而二分圖用途是來研究兩種不同類型的物件之間的關係，例如：有 M 個求職者和 N 個職位，每個申請人都有他們感興趣的工作子集，每個職位空缺只能接受一個申請人，並且只能為一個職位任命一個職位申請人。為求職者分配工作，讓盡可能多的求職者找到工作。

Hungarian Algorithm 的核心就是尋找增廣路徑，是一種用增廣路徑求二分圖最大匹配的算法，主要是為了在一個多項式時間內解決與二分圖匹配有關的問題，在 1955 年由美國數學家 Harold W. Kuhn 所開發並發表，他將此演算法命名為匈牙利演算法，是因為演算法很大一部分是基於以前匈牙利數學家 Dénes König 和 Jenő Egerváry 的早期作品建立起來的。James Raymond Munkres 在 1957 年審查該演算法時，發現它是(強)多項式的。此後該演算法被稱為 Kuhn–Munkres 演算法或 Munkres 分配演算法。

Edmonds Karp Algorithm 主要是將 Maximum Bipartite Matching 的問題轉換成最大流的問題，而最大流就是圖中最大的匹配數。

Edmonds Karp 演算法在 1970 年由 Yefim Dinitz 首次發表。

我們的專題將探討如何利用 Hungarian method 及 Edmonds-Karp algorithm 實作 maximum bipartite matching

二、文獻探討

(一) Bipartite graph：

根據文獻[1]的第四章節，Bipartite graph 的定義為圖中的每個點可以分成兩個互斥集合 V_0 和 V_1 ，使得所有邊的兩個端點，一個點屬於 V_0 集合，一個點屬於 V_1 集合。

(二) Bipartite matching：

根據文獻[1]的第四章節，Bipartite matching 的定義是圖 $G=(V, E)$ 是子圖 $G'=(V, E')$ ， $E' \subseteq E$ ，這樣沒有兩條邊 $e_1, e_2 \in E'$ 共享同一個頂點。

(三) Maximum Matching：

根據文獻[2]，令 $G=(V, E)$ 為有限無向連通圖，其頂點集合為 V 、邊的集合為 E 、匹配的集合為 M ，且 M 是 E 的子集，使得 M 中沒有兩條邊共用一個頂點，而 Maximum Matching 就是基數最大的匹配。

(四)增廣路徑(Augmenting path)：

根據文獻[3]的第二章節，增廣路徑是說起點和終點都是目前的 matching 所沒碰到的點，且中間一條邊不在目前的 matching 裡、一條邊在目前的 matching 裡、下一條邊又不在目前的 matching 裡，如此交錯的路徑，稱為增廣路徑。例如，Figure 1 中的 (v_5, u_2, v_1, u_3) 即為一條增廣路徑。

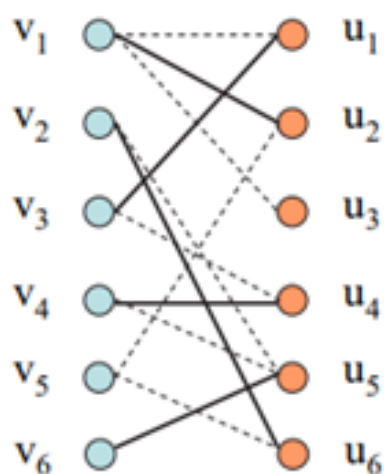


Figure 1

三、研究方法

3.1、初始架構 - 輸入設定

將一個分類好的二分圖輸入到程式中：

1. 首先輸入欲測試之演算法，輸入 1 為 Hungarian Method、輸入 2 為 Edmonds Karp(BFS)、輸入 3 為 Edmonds Karp(DFS)。
2. 接著分別輸入兩個頂點集合的個數，分別放入程式中已宣告的 m 和 n 中。
3. 最後使用 while 迴圈讓使用者輸入目前所有的配對邊，而我們透過先前宣告的 m 和 n 創造出兩個可以容納所有配對邊且不會 overflow 的 vector 來儲存此用者所輸入的配對邊。

3.2、初始架構 - 初始值設定與宣告

(1) Hungarian Algorithm:

`vector<int>girls:`

紀錄女生(V0 集合的頂點)和哪個男生(V1 集合的頂點)配對。

`int matches`: 紀錄最大的配對數。

`vector<bool>visited`:

紀錄男生(V_1 集合的頂點)是否被拜訪過。

(2) Edmonds Karp Algorithm:

`vector<vector<int>>grid`:

`grid[i][j]=1` 表示 V_0 集合的 i 和 V_1 集合的 j 是有路徑的。

`int source`: 代表源點 S 的編號。

`int sink`: 代表匯點 T 的編號。

`vector<bool>vis`: 記錄頂點是否已被拜訪過。

`vector<int>p`: 記錄頂點的父親。

3.3、方法流程 - Hungarian Algorithm

Hungarian algorithm 的核心就是尋找增廣路徑，是一種用增廣路徑求二分圖最大匹配的算法，而我們程式的執行流程如下，以 Figure 2 為例，其初始配對數為

2，且在 Figure 2 中原有配對邊的集合為(1, 6)和
(2, 7)：

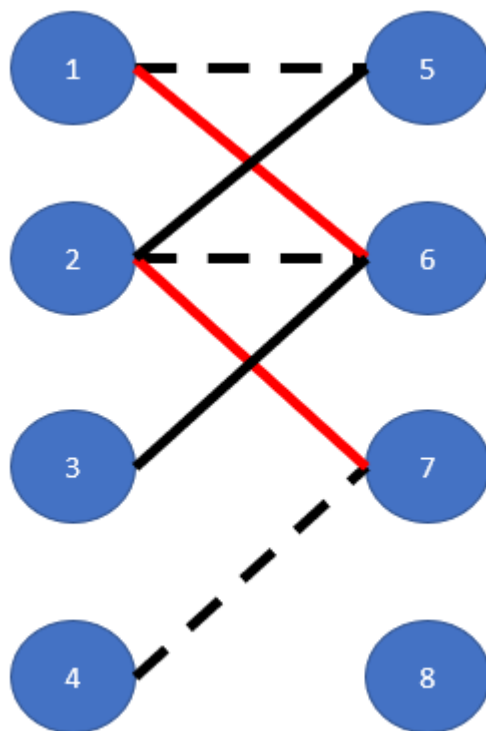


Figure 2

Step1：為了使配對數增加，我們繼續尋找有沒有其他更大的增廣路徑，而經過尋找後可以發現一條由虛線和紅線組成的增廣路徑，如 Figure 3 中由頂點 5、1、6、2、7、4 所形成的綠色 Path。

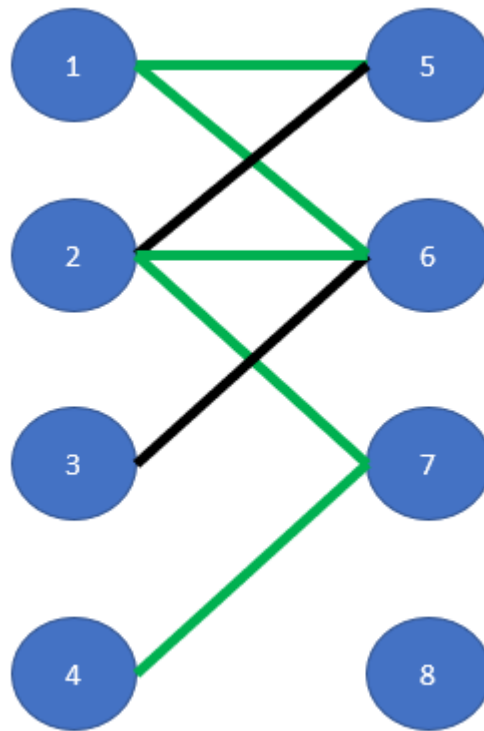


Figure 3

Step2：在增廣路徑中移除本來在配對裡面的邊，也就是將 $(1, 6)$ 和 $(2, 7)$ 從配對邊的集合中移除，並將新的一組配對 $(1, 5)$ 、 $(2, 6)$ 、 $(4, 7)$ 加入配對邊的集合中，如 Figure 4，因此成功將原配對數 2 擴大至 3，且此時已無法找到更大的增廣路徑，所以得到最大匹配數為 3。

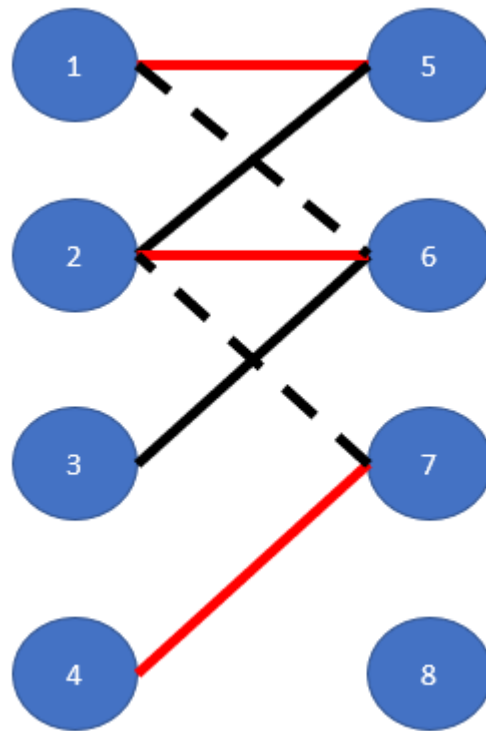


Figure 4

若還不是最大匹配數，則可以找到更大的增廣路徑，使配對數增大，因此可以依照此二步驟繼續執行，直到無法找到更大的增廣路徑為止，而此時可得到最大匹配數。

3.4、方法流程 - Edmonds Karp Algorithm

Edmonds Karp Algorithm 會將 Maximum Bipartite Matching 的問題轉換成最大流問題。程

式剛開始會建立源點 S 和匯點 T ， S 會連到 V_0 集合中的每個點， T 會連到 V_1 集合中的每個點(Figure 5)。

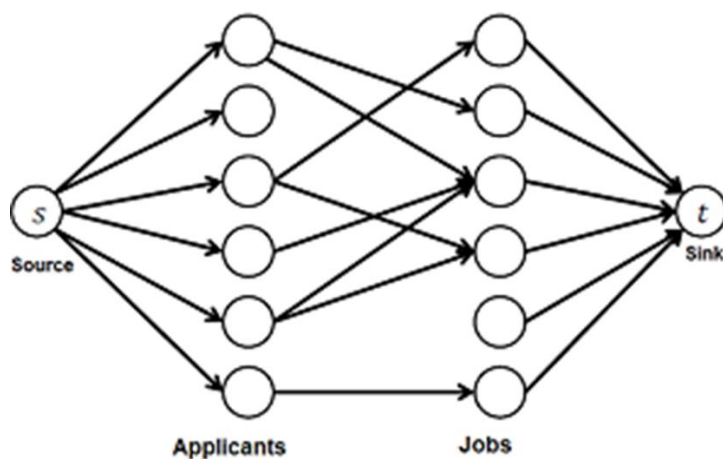


Figure 5

若將 Bipartite Graph 中的每條邊視為水管，則每條邊的最大容量會設成 1，且剛開始每條邊的流量都會是 0。Edmonds Karp Algorithm 中的增廣路徑的定義是，一條路徑，起點是 S ，終點是 T 。因此，在 Bipartite Graph 中，每條增廣路徑的最大容量就會是 3。程式剛開始會從源點使用 BFS(或 DFS)找到一條增廣路徑，當第一次到達匯點 T 後就停止搜索，再將增廣路徑上的每條邊的流量設成 1，並在增廣路徑上，建立反向邊，反向邊的流量都設成 0，反

向邊的容量都設成 1(如 Figure 6)。然後再繼續從源點 S 開始找增廣路徑，若不能再找到增廣路徑則得到最大流，而最大流即此圖最大匹配數。

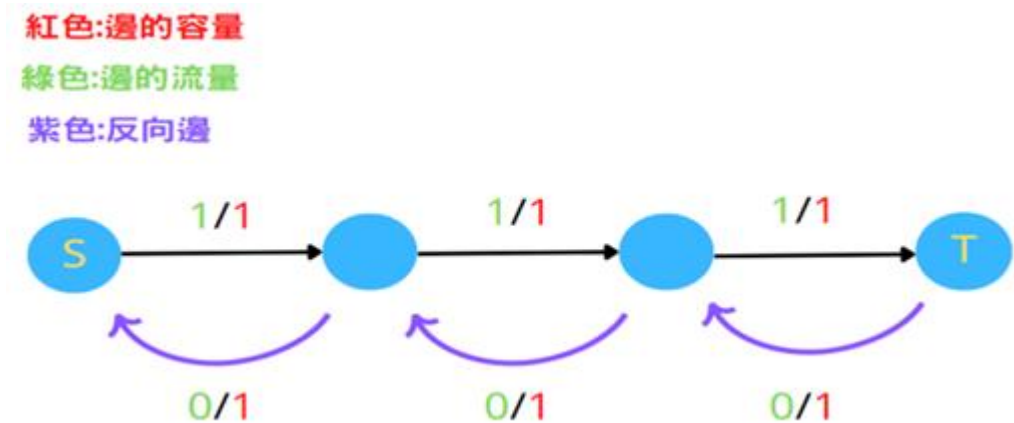


Figure 6

3.5、測資來源

我們的測資是透過程式讓電腦從 20 個頂點中隨機選取右邊點集合個數與左邊點集合個數，接著用兩層的 for 迴圈繼續讓電腦隨機選取配對邊個數與其欲配對之兩的頂點，之後將選取的結果寫入 input.txt，並透過 while 迴圈讓上述步驟執行 100 次，使我們可以在這 100 次中取得這 3 個演算法的平均，而其中因為重複之邊不影響實驗結果，因此我們在此程式中沒有特別去刪除重複之邊。

四、討論與分析

(一)為何可以透過 Hungarian Algorithm 找尋並置換增廣路徑來找到二分圖的最大匹配？

證明匈牙利演算法的正確性：

Theorem:

對所有 matching M ，若 M 非最大的配對集，則存在一個 M 的增廣路徑。(相當於：若配對集 M 沒有增廣路徑，則配對集 M 是最大的)

Pf:

現在有一個比 M 更大的配對集 M' 。

M' 一定存在，因為 M 不是最大的配對集。

設在 M 集合中但不在 M' 中的任意一個邊為 e 。

設在 M' 集合中但不在 M 中的任意一個邊為 e' 。

現在考慮一張圖 G ，此張圖的每個點最多只能碰到一個 e 及一個 e' ，否則會違反 matching 裡面的邊不能共用端點的規則，因此圖 G 中每個點的 Degree 會是小於等於 2。

此證明引用一個「事實」：

當每個點的 Degree 最大為 2 時，可以將圖 G 切割成 cycle 或 path。

利用上面的「事實」，因此可以將圖 G 切割成 cycle 或 path(如 Figure 7)。

cycle 的長度一定是偶數，否則會違反 matching 裡面的邊不能共用端點的規則。因為圖 G 中每個點最多只能碰到一個 e 及一個 e' ，因此 cycle 中的 e 及 e' 的個數相同。

由於 M' 集合比 M 集合大，因此 e' 的個數比 e 的個數多。

因為 cycle 中的 e 及 e' 的個數相同，因此圖 G 中一定會有一條 path Q 存在，且此條 path Q 中 e' 的個數比 e 的個數多。

由於圖 G 中每個點最多只能碰到一個 e 及一個 e' ，且 path Q 中 e' 的個數比 e 的個數多，因此 path Q 中的兩端點 A 、 B 連到的邊一定都是 e' ，則此條 path Q 滿足增廣路徑的定義，因此 path Q 為一條增廣路徑。

證明 path Q 一定是一條增廣路徑:(反證法)

設 path Q 中的端點為 A、B。

若 A 點能向外再延伸出一條邊連到新的端點 C，則此條邊必存在 M 集合中，因為 path Q 中的兩端點向內連到的邊一定都是 e' ，而且圖 G 中每個點最多只能碰到一個 e 及一個 e' 。設此條為新的 path R(端點是 C、B)，矛盾(因為 path Q 的兩端點是 A、B，與原假設不符)。同理可證明 path Q 中的另一個端點 B。

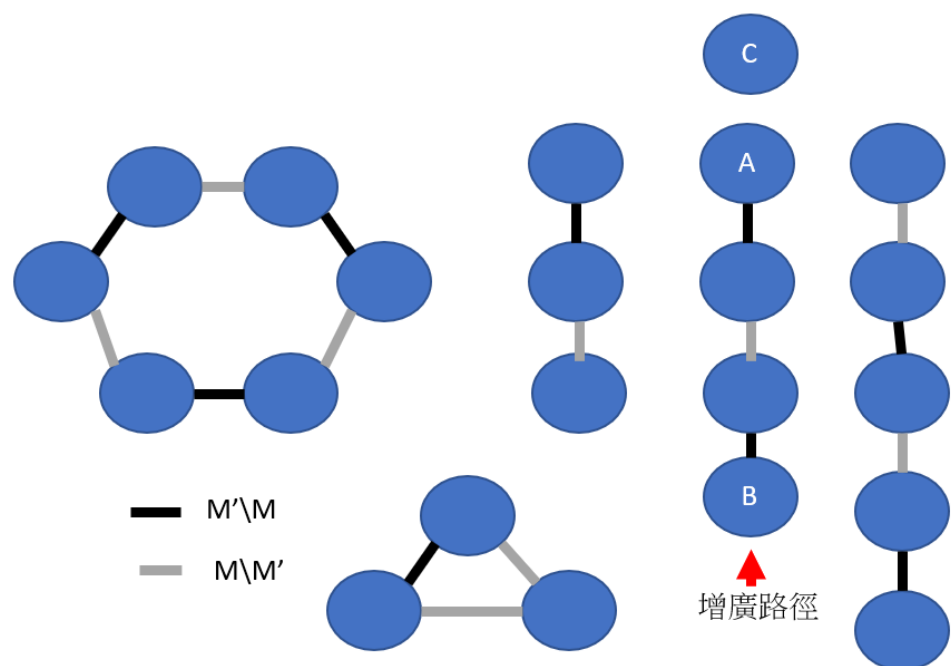


Figure 7

(二)為何測資的產生方式是用隨機的？

因為 Maximum Bipartite Matching 問題的測資，沒有什麼特別的規則，因此程式比賽的命題委員們才會覺得困擾，所以我們才用隨機的方式產生測資。

(三)Hungarian algorithm 的應用？

有 M 個考生和 N 個科目，每位考生都需要看資演、計系、數學，但是由於可以準備考試的時間不足，因此讓每位考生只準備一個科目，並讓每位考生互相教學沒看的科目，以此增進效率。下表的圖是每位考生看各科所需要的時間，以此表決定各考生該看哪科，可以讓總體所花費的時間成本最小。

科目 人名	資結演算法	計組 OS	線代離散
元智	2 hr	3 hr	3 hr
陽交	3 hr	2 hr	3 hr
清華	2 hr	3 hr	2 hr

當把匈牙利方法應用於上面的表格時，會得到最低的時間成本也就是最有效率的方法為 6hr，讓元智讀資演、讓陽交讀計系、讓清華讀數學，就可以達到這個時間成本。

五、結論

從 Figure 8 中可以發現，這三種演算法經由程式所隨機產生的 100 筆測資後，接著讓程式在取得每個測資的時間並取平均值做比較，因此根據 Figure 8，可以發現 Hungarian algorithm 的平均時間是最短的，而 Edmonds Karp algorithm (BFS) 的平均時間是最長的，Edmonds Karp algorithm (DFS) 的平均時間是中間值。

演算法	平均時間(ms)
Hungarian algorithm	0.95227 <u>ms</u>
Edmonds Karp algorithm (DFS)	2.60678 <u>ms</u>
Edmonds Karp algorithm (BFS)	3.10415 <u>ms</u>

Figure 8

- 實際成果 - 測資產生

```

11 string filename("input.txt");
12 fstream file_out;
13
14 file_out.open(filename, std::ios::out);
15 if (!file_out.is_open()) {
16     cout << "failed to open\n";
17 }
18 /*else {
19     file_out << "Some random data\n";
20     cout << "Done Writing!\n";
21 }*/
22 srand(time(NULL));
23 int time_t = 0;
24
25 while (time_t < 100)
26 {
27     int num = 20;
28     int left = rand();
29     int right = rand();
30     left = (left % num) + 1;
31     right = (right % num) + 1;
32 }

```

輸出: C:\Users\User\Desktop\元智資工\大學專題(Maximum bipartite matching)\專題海報與code\專題測資\Debug\專題測資.exe (處理序 24308) 已結束, 出現代碼 0。

- 實際成果 - Hungarian method

```

220
221 if (choice == 4)
222     break;
223
224 //cout << "請輸入選擇: ";
225 file >> temp;
226 string_to_int << temp;
227 string_to_int << temp;
228
229 //cout << endl;
230 //cout << "請輸入選擇: ";
231 file >> temp;
232 string_to_int << temp;
233 string_to_int << temp;
234
235 //cout << endl;
236 //cout << "請輸入選擇: ";
237 file >> temp;
238 string_to_int << temp;
239 string_to_int << temp;
240
241 double res = 0;
242 cout << "average time : 0.952272 ms\n";
243 return 0;

```

輸出: C:\Users\User\Desktop\元智資工\大學專題(Maximum bipartite matching)\專題海報與code\專題測資\Debug\專題測資.exe (處理序 8940) 已結束, 出現代碼 0。

- 實際成果 - Edmonds Karp(BFS)

```

220
221 if (choice == 4)
222     break;
223
224 //cout << "請輸入 Max matches: 9
225 file >> temp; Edmonds Karp(BFS)演算法的執行時間:2.5627 ms
226 string_to_int << "Max matches: 13
227 string_to_int << "Edmonds Karp(BFS)演算法的執行時間:2.3308 ms
228 //cout << endl; Edmonds Karp(BFS)演算法的執行時間:6.7849 ms
229 //cout << "Max matches: 7
230 //cout << "Edmonds Karp(BFS)演算法的執行時間:2.4464 ms
231 //cout << "請輸入 Max matches: 3
232 file >> temp; Edmonds Karp(BFS)演算法的執行時間:0.9682 ms
233 string_to_int << "Max matches: 6
234 string_to_int << "Edmonds Karp(BFS)演算法的執行時間:1.1335 ms
235 //cout << endl; Edmonds Karp(BFS)演算法的執行時間:0.3791 ms
236 //cout << "請輸入 Max matches: 7
237 //cout << "Edmonds Karp(BFS)演算法的執行時間:6.1079 ms
238 if (file.eof())
239 {
240     double res = C:\Users\User\Desktop\元智資工\大學專題(Maximum bipartite matching)\專題海報與code\專題\Debug\專題.exe (處理序 12068) 已
241     cout << "average time : 3.10415 ms
242     return 0;
243 }
  
```

- 實際成果 - Edmonds Karp(DFS)

```

220
221 if (choice == 4)
222     break;
223
224 //cout << "Max matches: 9
225 file >> temp; Edmonds Karp(DFS)演算法的執行時間:2.1576 ms
226 string_to_int << "Max matches: 13
227 string_to_int << "Edmonds Karp(DFS)演算法的執行時間:1.6265 ms
228 //cout << endl; Edmonds Karp(DFS)演算法的執行時間:5.7142 ms
229 //cout << "Max matches: 7
230 //cout << "Edmonds Karp(DFS)演算法的執行時間:1.9107 ms
231 //cout << "Max matches: 3
232 file >> temp; Edmonds Karp(DFS)演算法的執行時間:0.8513 ms
233 string_to_int << "Max matches: 6
234 string_to_int << "Edmonds Karp(DFS)演算法的執行時間:0.8363 ms
235 //cout << endl; Edmonds Karp(DFS)演算法的執行時間:0.3871 ms
236 //cout << "Max matches: 7
237 //cout << "Edmonds Karp(DFS)演算法的執行時間:3.3627 ms
238 if (file.eof())
239 {
240     double res = C:\Users\User\Desktop\元智資工\大學專題(Maximum bipartite matching)\專題海報與code\專題\Debug\專題.exe (處理序 25480) 已
241     cout << "average time : 2.60678 ms
242     return 0;
243 }
  
```

六、References

- [1] Bellur, Umesh, and Roshan Kulkarni. "Improved matchmaking algorithm for semantic web services based on bipartite graph matching." IEEE international conference on web services (ICWS 2007). IEEE, 2007.
- [2] Giel, Oliver, and Ingo Wegener. "Evolutionary algorithms and the maximum matching problem." Annual Symposium on Theoretical Aspects of Computer Science. Springer, Berlin, Heidelberg, 2003.
- [3] Mills-Tettey, G. Ayorkor, Anthony Stentz, and M. Bernardine Dias. "The dynamic hungarian algorithm for the assignment problem with changing costs." Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-07-27 (2007).