# 12 隨機演算法

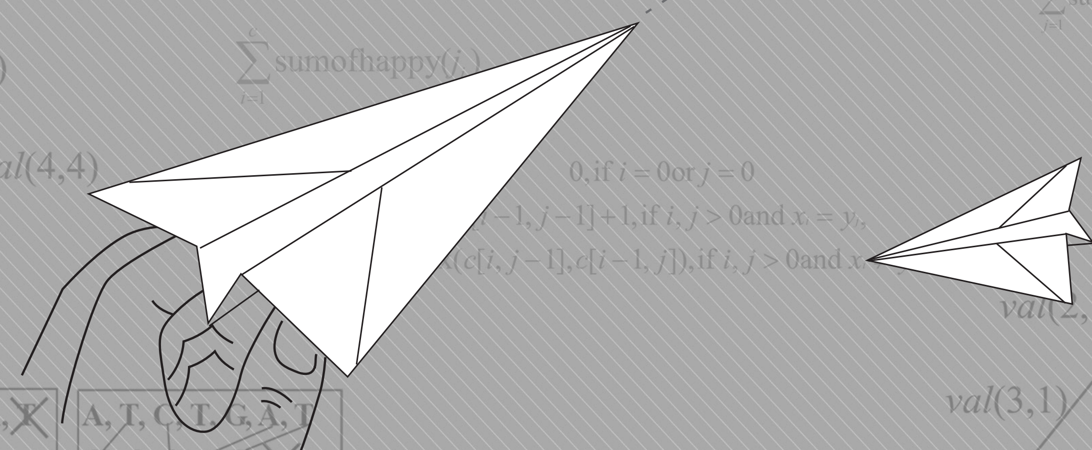
**章節大綱**



虛竹慈悲之心大動，心知要解段延慶的魔障，須從棋局入手，只是棋藝低淺，要說解開這局複雜無比的棋中難題，當真是想也不敢想。眼見段延慶雙目呆呆的凝視棋局，危機生于頃刻，突然間靈機一動：「我解不開棋局，但搗亂一番，卻是容易，只需他心神一分，便有救了。既無棋局，何來勝敗？」，便道：「我來解這棋局。」快步走上前去，從棋盒中取過一枚白子，閉了眼睛，隨手放在棋局之上。

他雙眼還沒睜開，只聽得蘇星河怒聲斥道：「胡鬧，胡鬧，你自填一氣，自己殺死一塊白棋，哪有這等下棋的法子？」，虛竹睜眼一看，不禁滿臉通紅。原來自己閉著眼睛瞎放一子，竟放在一塊已被黑棋圍得密不通風的白棋之中。這大塊白棋本來尚有一氣，雖然黑棋隨時可將之吃淨，但只要對方一時無暇去吃，總還有一線生機，苦苦掙扎，全憑于此。現下他自己將自己的白棋吃了，棋道之中，從無這等自殺的行徑。這白棋一死，白方眼看是全軍覆沒了。鳩摩智、慕容復、段譽等人見了，都不禁哈哈大笑。

金庸 天龍八部



# 12.1 何謂隨機演算法？

「什麼是隨機演算法(randomized algorithms)？」

**簡言之：「一個演算法中含有隨機決定的步驟。」**

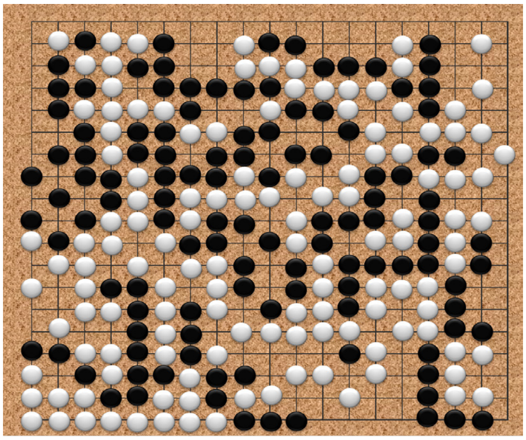
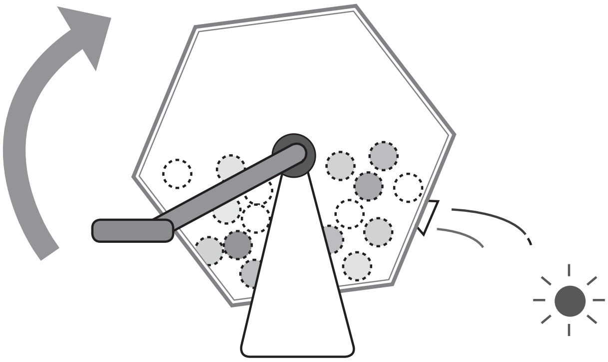


圖 12.1　天龍八部電視劇中的珍瓏棋局
(資料來源：一元一國學網 www.yiyuanyi.org)

一般的演算法其步驟都是經過小心設計，並且深具目的性。然而，隨機演算法恰恰反其道而行，在某一些步驟中，竟然含有隨機決定的指令。此舉有如金庸小說中的虛竹，因為胡亂下一子後，竟然意外地解開了古今難破的珍瓏棋局。隨機演算法的存在，彷彿是在告訴人們「道法自然」有時比「汲汲經營」有效率。



# 12.2 隨機快速排序法

隨機演算法的第一個例子，就是我們已經熟悉的快速排序法(請參閱第 2 章)。

「如何避免快速排序法的最差狀況？」

「讓切割元素的值儘量靠近中間值。如此可以讓切割更平均，並讓快速排序法執行加快。」

一個極簡單的***隨機快速排序法***(randomized quicksort)就是在執行切割前，先將矩陣的第一個數和其它任意一數(由亂數決定)作對調。如此可大幅降低選中最大值或最小值為切割元素的機會，尤其是當資料輸入時就大致排序好的狀況。隨機快速排序法如表 12.1 所示。

**表 12.1 隨機快速排序法**

| | |
|---|---|
| **輸入** | $a[p], \cdots, a[q]$ |
| **輸出** | $a[p] \leqq a[p+1] \cdots \leqq a[q]$ |
| **步驟** | (見下方程式) |

```
Algorithm rquicksort(p, q)
{
if (p<q) then
    {if ((q-p) >5 then
        interchange (a, (Random() mod (q-p+1))+p, p);
          /*將矩陣 a 的第一個數和其它任意一數作對調*/
        j:= partition(a, p, q+1);
        rquicksort(p, j-1);
        rquicksort(j+1, q);
    }
}
```

隨機快速排序法的期望執行時間，可被證明為 $O(n \log n)$，比原先的快速排序法的最差時間複雜度 $O(n^2)$快。

# 12.3 質數測試

第二個例子是質數測試(prime number testing)。

**表 12.2 質數測試**

| | |
|---|---|
| **問題** | 在通訊安全的應用中，常需要產生一個十分大的質數(即除了 1 和本身外，沒有其他正因數)。但是使用一般的方法來測試一個正整數是否為質數，有時不夠快速。請設計一個有效率的演算法來解決此問題 |
| **輸入** | 正整數 $n$(>2)<br>13 |
| **輸出** | 回答此正整數 $n$ 是否為質數？<br>是 |

測試一個正整數 $n$ 是否為質數的簡單方法是，試著將 $n$ 除以 2，3，…，$\left\lfloor \sqrt{n} \right\rfloor$。若無任一數可以整除，則 $n$ 為質數；若存在任一數可以整除，則 $n$ 不是質數。這個方法看似不錯，但是當 $n$ 是一個大數時，將需要冗長的計算。

為了設計一個有效率的質數測試演算法，需要認識以下的性質。

**表 12.3 費馬定理**

**Fermat's Theorem**

如果 $p$ 是質數，則 $a^{p-1}=1 \pmod{n}$，針對所有的 $a \in \{1, 2, \cdots, p-1\}$。

換言之，如果找到一個 $a \in \{1, 2, \cdots, p\text{-}1\}$，使得 $a^{p-1} \neq 1 (\mathrm{mod}\ n)$，則 $n$ 必不是質數。令人驚奇的是，當 $a^{p-1}=1(\mathrm{mod}\ n)$時，則 $n$ 為質數的機率竟然也十分高。因此，這個條件(即 $a^{p-1}=1(\mathrm{mod}\ n)$否?)，就成為檢測質數的幾乎完美的方法。

**表 12.4 隨機質數測試演算法**

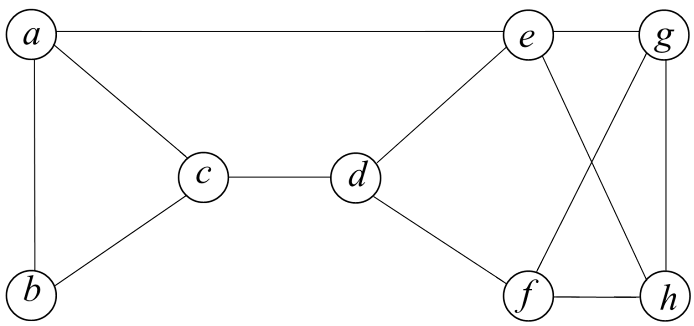
| | |
|---|---|
| **輸入** | 正整數 *n*(>2)，正整數 *s* |
| **輸出** | 回答此正整數 *n* 是否為質數 |
| **步驟** | <pre>Algorithm primality_test(n,s)<br>{<br>Step 1: for j=1 to s do         //重複執行下列步驟 s 次<br>  {<br>  1.1: a=random(1, n-1);       //自 1 到 n-1 的整數中隨機選出一數<br>  1.2: 如果 a^(n-1)≠1(mod n)，則回傳 n 不是質數;       //確定<br>  }<br>Step 2: 回傳 n 是質數;                              //不確定<br>}</pre> |

當隨機質數測試演算法，判斷輸入值 $n$ 不是質數時，必是正確的。但是，當它判斷輸入值 $n$ 是質數時，可以被證明其錯誤機率不高於 $2^{-s}$。即當 $s$ 足夠大時，此演算法的正確性，是值得被期待的。

# 12.4 最小線切割

最後的例子是最小線切割 (minimum edge cut)。

**表 12.5 最小線切割問題**

| | |
|---|---|
| **問題** | 軍方人員準備截斷敵軍某網路中的線路，以阻斷其通訊。為求時效，希望截斷最少的網路線，以切割整個網路並使之斷裂不連通。請您設計一個演算法，找出這樣的網路線 |
| **輸入** | 一個圖 $G=(V, E)$代表現存的網路。每個點代表網路上的節點，而節點間的線，代表連接兩節點的網路線  |
| **輸出** | 一個線集合 $E$ 的最小子集合 $E'$，使得除去此子集合 $E'$ 後的圖 $G'=(V, E-E')$是不連接的 $\{(a, e), (c, d)\}$ |

一個圖上的***線切割***(edge cut)為線集合的子集合，若將此子集合移除，將導致此圖不連接。最小線切割，即是所有線切割中，擁有最少線者。例如，表 12.5 中的圖，$\{(a, e), (c, d)\}$或$\{(b, a), (b, c)\}$就是一個最小線切割。

以下，我們將設計一個隨機演算法來找出最小線切割。此方法十分簡單，即隨機地重複地找到多組線切割，並挑其中最小的線切割當作輸出。

隨機地產生一個線切割的方法也十分簡單，說明如下：首先，任意(隨機地)自圖中挑一條線，隨後將此線和其兩端點***壓擠***(contract)成一個點，但保留連接兩端點的其他連線。圖 12.2 顯示將線$(c, d)$和其兩端點壓擠後的新圖。

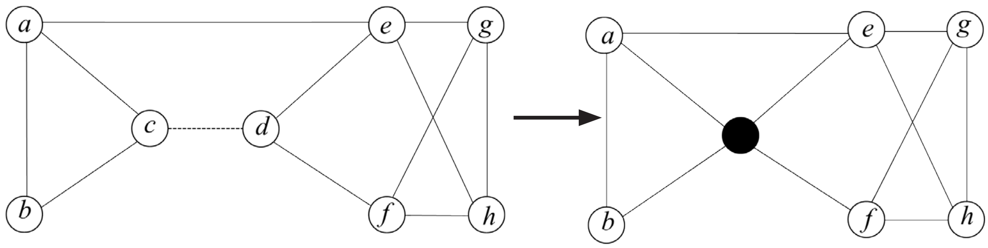
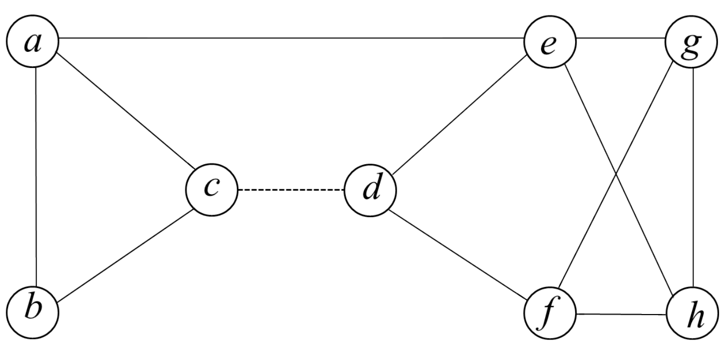


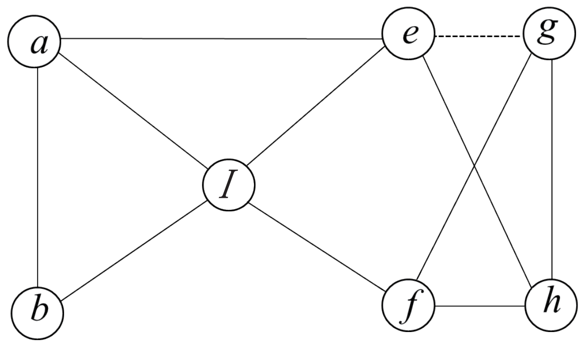

圖 12.2　壓擠線(*c*, *d*)和其兩端點後，以一個新點取代此線

重複以上的動作，直到剩下兩點為止。注意，此時所有連接此兩點的線便是一個線切割(雖然有可能不是最小線切割)。例如，圖 12.3 演示找到一個線切割的過程。倘若還原到輸入圖，所找到的線切割是{($e$, $h$), ($f$, $h$), ($g$, $h$)}。注意，此範例的最小線切割含有的線應為兩條，如{($a$, $e$), ($c$, $d$)}或{($a$, $b$), ($b$, $c$)}。故此次找到的線切割並非最小。

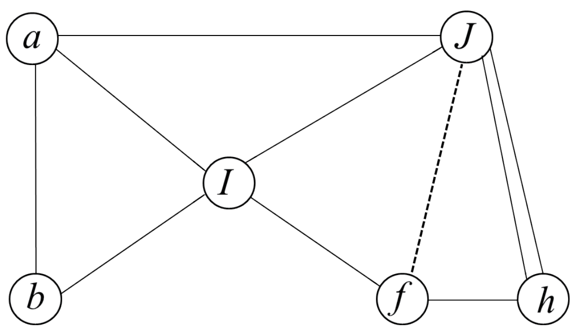
乍看之下，這個方法好像十分容易出錯，但是只要重覆的次數夠多，找出正確解(即最小線切割)的機率便增加。這種方法很像是用亂槍打鳥，只要射擊的次數夠多，擊中的範圍夠廣，則擊中空中小鳥的機會便會增加。



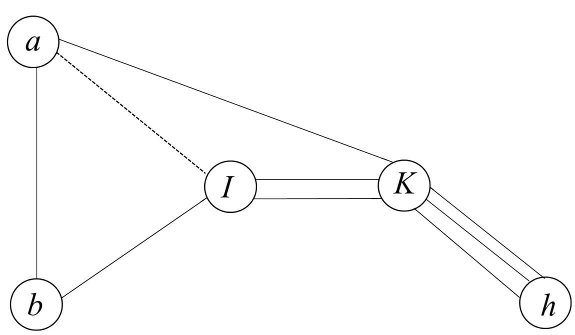

選擇(*c*, *d*)



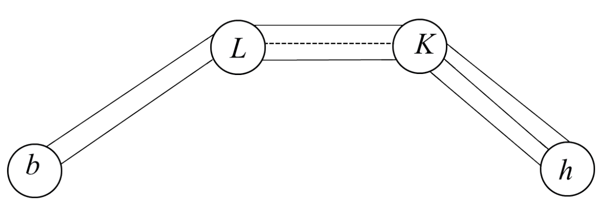

$(c, d)$被擠壓成 $I$
選擇$(e, g)$



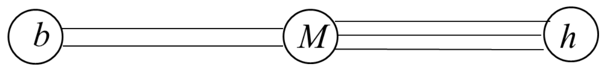

$(e, g)$被擠壓成 $J$
選擇$(f, J)$



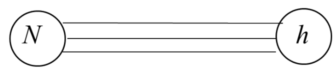

$(f, J)$被擠壓成 $K$
選擇$(a, I)$




$(a, I)$被擠壓成 $L$
選擇$(L, K)$




$(L, K)$被擠壓成 $M$
選擇$(b, M)$




$(b, M)$被擠壓成 $N$

圖 12.3　隨機地產生一個線切割的方法。當還原回到起始圖時，所找到的線切割是$\{(e, h), (f, h), (g, h)\}$

此隨機演算法乃是利用上述步驟，產生足夠多組的線切割後，並挑其中最小的線切割，當作輸出。隨機最小線切割演算法列於表 12.6。

**表 12.6 隨機最小線切割演算法**

<table>
<tr><td>輸入</td><td>一個連接圖 $G=(V, E)$及一個正整數 $s$</td></tr>
<tr><td>輸出</td><td>最小線切割 $C$(正確率大於 $1-(1-2/n^2)^s$)</td></tr>
<tr><td>步驟</td><td><pre>
Algorithm minimum_edge_cut(G, s)
{
Step 1: for j=1 to s do   //重複執行下列步驟 s 次，以找到多個線切割
  {
  1.1: V'=V; E'=E; C=E;
  1.2: 當 V'>2 則 do
       {
       隨機地自線集合 E'中選擇一線 e=(u, v);
       擠壓 e 成一個新點，並且更新 V'和 E';
       }
  1.3: 令 C'儲存所有連接最後兩點的線;     //找到一個新的線切割
  1.4: 當|C'|<|C|，使用 C'取代 C;    //找到更小的線切割
  }
Step 2: 輸出 C;
}
</pre></td></tr>
</table>

**「這個演算法找到最小線切割的機率有多高？」**

「看起來好像不太高，但應該和 $s$ 有關。」

**「有什麼關係？」**

「$s$ 的值愈高，找到的線切割愈多，會碰到最小線切割的機會也就愈大。」

**「這樣 $s$ 該設定多大才夠呢？」**

「這個…」

---

以下討論隨機最小線切割演算法，正確找到最小線切割的機率。首先，令最小線切割 $C$ 一共有 $k$ 條線。則圖上的每個點至少會有 $k$ 個鄰居；否則必有一點和其鄰居的連線，就形成一個比 $k$ 少條的線切割(出現矛盾)。因此，此圖至少擁有$(nk)/2$ 條線，此處 $n$ 為圖中點的個數。

我們希望計算出，在隨機選擇線(並擠壓此線)的過程中，最小線切割 $C$ 中的 $k$ 條線都不被選中的機率，即最後找到最小線切割的機率。注意隨機選擇線並擠壓此線的動作，共需要 $n$-2 個步驟；因為每做一次步驟(選擇線並擠壓此線)後，剩下的圖便少一個點(如圖 12.3 所示)。

因此，我們想計算在 $n$-2 次隨機擠壓過程中，不會選中最小線切割 $C$ 的機率。

第一次隨機擠壓的過程中，不會選中最小線切割 $C$ 的機率：

```
Pr[E1]等於 1-(最小線切割 C 的線個數)/(所有線的個數)=1-k/(所有線的個數)
```

因為此圖至少擁有$(nk)/2$ 條線，故：

```
Pr(E1)≧1-(k/(nk/2))=1-2/n
```

類似地，第二次隨機擠壓的過程中，不會選中最小線切割 $C$ 的機率 ：

```
Pr(E2)≧1-(k/((n-1)k/2))=1-2/(n-1)
```

類似地，我們可得

```
Pr(E3)≧1-(k/((n-2)k/2))=1-2/(n-2)
 ⋮
Pr(En-2)≧1-(k/((3)k/2))=1-2/(3)
```

此處 $E_i$ 代表在第 $i$ 步驟時並未挑中最小線切割 $C$ 中的線，而 $\Pr(E_i)$代表其機率(1<=$i$<=$n$-2)。

最後，在 $n$-2 次隨機擠壓(一條線)的過程中，都不會選中最小線切割 $C$ 的機率將大於或等於

$$(1-2/n)\times(1-2/(n-1))\times\ldots\times(1-2/3)$$
$$=\frac{n-2}{n}\times\frac{n-3}{n-1}\times\ldots\times\frac{2}{4}\times\frac{1}{3}=2/(n\times(n-1))\geq 2/n^2$$

隨機最小線切割演算法執行上述動作 $s$ 次，因此都得不到最小線切割 $C$ 的機率將小於$(1-2/n^2)^s$；即得到最小線切割 $C$ 的機率將大於 $1-(1-2/n^2)^s$。若選取較大的 $s$ 值(如 $n^2/2$ 的倍數)時，則此演算法出錯的機率，將降低到可以被接受的程度。

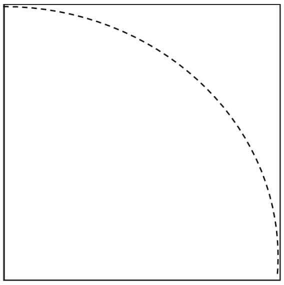
## 12.5 隨機演算法技巧

隨機演算法的優點在於其簡潔及效率。一般分成兩類：***拉斯維加斯演算法***(Las Vegas algorithm) 和***蒙地卡羅演算法*** (Monte Carlo algorithm)。

拉斯維加斯演算法總是輸出正確的答案，隨機的步驟影響的是執行的時間，如隨機快速排序法。相對地，蒙地卡羅演算法有時會輸出錯誤的答案，但是獨立地重複執行多次，會將錯誤的機率降低，如隨機質數測試演算法和隨機最小線切割演算法。

## 學習評量

1. 請設計一個蒙地卡羅演算法(Monte Carlo algorithm)，來計算以下 1/4 圓的面積。您的演算法，將在此正方形中，均勻地產生大量的點後，利用落在 1/4 圓內點的比例，來計算此 1/4 圓的面積。



| | |
|---|---|
| **輸入** | |
| 10 | (正方形的寬 *k*) |
| **輸出** | |
| 78.53 | (1/4 圓的面積) |

2. 請設計一個隨機快速排序法(randomized quicksort)，並且與原來的快速排序法進行比較。

| | |
|---|---|
| **輸入** | |
| 5 | (被排序的數字個數) |
| 65 | (以下是被排序的數字) |
| 50 | |
| 55 | |
| 45 | |
| **輸出** | |
| 45 50 55 65 | (排序好的數列) |
| 0.0012 秒 | (排序所需的時間) |

3. **質數比率**：請利用費馬定理，設計一個隨機質數測試演算法。並請嘗試計算任意一個連續整數的範圍中，有多少比率是質數。

| | |
|---|---|
| **輸入** | |
| 1 100 | ($m$, $n$ 代表連續整數的範圍 $m$~$n$) |
| **輸出** | |
| 25 | (在範圍 $m$~$n$ 中，所有質數個數) |
| 0.25 | (出現質數的比率) |

# 參考文獻


1. Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.
2. J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*, North-Holland, 1976.
3. T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, The MIT Press, 1998.
4. Shimon Even, *Graph Algorithms*, Cambridge University Press, 2011.
5. Lestor R. Ford, Jr., and D. R. Fulkerson, *Flows in Networks*, Princeton University Press, 1962.
6. Michael R. Garey and David S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, San Francisco: W. H. Freeman & Co., 1979.
7. R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete Mathematics*, Addison-Wesley, 1994.
8. Dorit S. Hochbaum, editor, *Approximation Algorithms for NP-hard Problems*, PWS Publishing Company, 1997.
9. Ellis Horowitz and Sartaj Sahni, *Fundamentals of Computer Algorithms*, Computer Science Press, 1978.
10. Ellis Horowitz and Sartaj Sahni, *Fundamentals of Data Structures*, Computer Science Press, Maryland, 1976.

11. Eugene L. Lawler, *Combinatorial Optimization*, Holt, Rinehart, and Winston, 1976.

12. R. C. T. Lee, R. C. Chang, S. S. Tseng, and Y. T. Tsai, *Introduction to the Design and Analysis of Algorithms*, Flag Publishing, 2001.

13. C. L. Liu, *Introduction to Combinatorial Mathematics*, McGraw-Hill, 1968.

14. Udi Manber, *Introduction to Algorithms: A Creative Approach*, Addison-Wesley, 1989.

15. G. Ploya, *How to Solve It*. Garden City, NY: Doubleday, 1957.

16. Artaj Sahni, *Concepts in Discrete Mathematics*, Camelot Pub. Co., 1981.

17. Robert Sedgewick, *Algorithms*, 2nd ed. Addison-Wesley, 1988.

18. M. N. S. Swamy and K. Thulasiraman, *Graphs, Networks, and Algorithms*, John Wiley and Sons, Inc, 1981.

19. Robert Endre Tarjan, *Data Structures and Network Algorithms*, Society for Industrial and Applied Mathematics, 1983.

20. http://mathworld.wolfram.com/topics/GraphTheory.html

21. 金庸，*笑傲江湖*，台北：遠流出版公司，1994年4月二版。

22. 金庸，*天龍八部*，台北：遠流出版公司，1987年4月二版。