臺灣二 OO 四年國際科學展覽會

科　　別：數學科

作品名稱：鬼角圖的數學原理

學　　校：國立臺南第一高級中學

作　　者：林義軒、蘇億城

作 者 簡 介

作者一：林義軒

我是林義軒，今年 16 歲，目前就讀台南一中 2 年級。興趣是看小說、聽音樂、上 BBS。在學校沒有哪一科表現特別突出，最有興趣的是物理和數學，此次參加科展也是因爲如此。目前處於少年維特的憂鬱時期，認爲快樂是沒有意義的東西，不喜歡空談，只要有符合現實的理想就會去完成；雖然我害怕失敗，容易沮喪，有時候還會半途而廢，但我總是認爲：儘管在怎麼軟弱，也總要有堅強的一面。

作者二：蘇億城

我的名字叫蘇億城，目前就讀台南一中二年級。我喜歡閱讀、玩電腦和用數學難題操自己的腦袋，不喜歡坐在書桌前背課文。學習一向眼隨意走、隨興所至，埋首在數理研究，並不太在意課業，所以學校成績不甚理想，尤其是國文及英文，老是在六七字頭打轉。

這次參加數學科展剛開始的動機純粹是因爲公假和應付學分，不過後來的進展實在是大出我的意料之外，於是我也花了不少時間在上面。

Abstract

We can prove 鬼腳圖 have an one-to-one characteristic; it is mean that you can not design a 鬼腳圖 which will make two starting point to the same end. We also can prove you can design any 鬼腳圖 you want; you can predict a result, and you can design a 鬼腳圖 which suit the result, no matter what the result it is.

We can design any 鬼腳圖 we want, but it possibly becomes very big and complicated. We develop a method to make it become briefer. According to the method, we make a function that can design the 鬼腳圖 you want in a very short time. You predict a result in computer, and the function will design a 鬼腳圖 which suit the result, and it will be the briefest.

摘　　要

吾人已經可以證明鬼腳圖具備一對一的性質，意思就是：不可能從兩個起點開始畫線，最後到同一個終點上。吾人亦證明：鬼腳圖的結果沒有限定：同一組初始條件可以轉換成任何一組結果。而同一組結果也有許多種不同的畫法，顯示鬼腳圖的畫法不具唯一性。即使如此，畫出來的鬼腳圖可能過於複雜，於是吾人又發展出簡化鬼腳圖的方法，可畫出較簡潔的鬼腳圖。吾人並根據這種化簡方式編出一套程式，只要將欲得的結果輸入，電腦就可以畫出最簡潔的鬼腳圖。

一、研究動機

鬼腳圖是很多人都玩過的遊戲，他的有趣之處在於做法簡單，但具有很多的可能性，看似複雜，但其中似乎隱藏著極為簡潔的規律形式。本研究的目的，正是要試圖找出這規律性。

二、研究目的

1. 研究鬼腳圖是否具一對一性
2. 研究鬼腳圖的交換是否具不限定性
3. 研究鬼腳圖的畫法是否具不唯一性

4. 研究鬼腳圖的簡化

三、研究步驟

運用直觀想法推理, 證明

運用數學符號表達證明方式

一時不能證明之處理運用電腦跑結果

四、研究結果：

（一） 鬼腳圖的數學原理

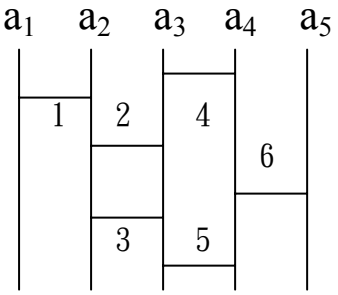
鬼腳圖是一種交換的概念，當我們在兩條線之間畫一條橫線時，其實就是把兩條線上原本的數據對調。所以，假設我們在第 k 條線 L_k 和第 k+1 條線 L_{k+1} 之間畫一條橫線時，則此橫線可以表示為（L_k，L_{k+1}），我們將（L_k，L_{k+1}）定義為：將原本在 L_k 上的數據和原本在 L_{k+1} 的數據對調。所以，若欲將原數據群 a_1、a_2、a_3、…….a_n 改為 b_1、b_2、b_3、…….b_n，則可以表示為函數 f: $\{a_1,a_2,a_3,\ldots\ldots..a_n\}\rightarrow \{b_1,b_2,b_3\ldots\ldots...b_n\}$。而 $\{b_1,b_2,b_3\ldots\ldots b_n\}=\{a_1,a_2,a_3,\ldots\ldots\ldots a_n\}$。

（二）遊戲規則：

現有 n 個數據, 以 a_1、a_2、a_3、…….a_n 表示。

1. 在紙上畫出 n 條平行線段，將 a_1、a_2、a_3、…….a_n 依序排在每條線段頂端。
   定義：a_1 所在的線段稱為 l_1，a_2 所在的線段稱為 l_2……以此類推。
2. 在相鄰的平行線段間任意畫上橫線，但是任兩條線皆不可等高。
   定義：L_1 和 L_2、之間的橫線表示為（L_1，L_2），以此類推……
   又：令整個圖中最高的一條橫線稱為第 1 位階，第二高的稱為第 2 位階……以此類推。
3. 由 a_1 開始順著 L_1 走，走到橫線（L_1，L_2）時就順著橫線走到 L_2，若是再碰到橫線（L_2，L_3）就在順著橫線走到 L_3……以此類推……其餘 a_2、a_3、…….a_n 均照此方法一一進行。
   定義：橫線（L_k，L_{k+1}）為一交換群概念，意為將 L_k 上的數據 a_p 置換至 L_{k+1} 上，且將 L_{k+1} 上的數據 a_q 置換至 L_k 上。
4. 每個數據 a_h 都需走到線段另一端為止，才能算完成

5. 範例：

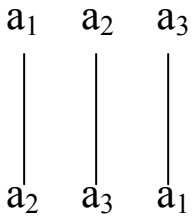


則橫線 4 位階為 1，記作 1(L_3，L_4)
橫線 1 位階為 2，記作 2(L_1，L_2)
橫線 2 位階為 3，記作 3(L_2，L_3)
橫線 6 位階為 4，記作 4(L_4，L_5)
橫線 3 位階為 5，記作 5(L_2，L_3)
橫線 5 位階為 6，記作 6(L_3，L_4)

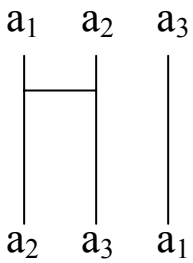
（三）鬼腳圖的逆推：

若現在已知鬼腳圖的答案，則要如何畫出這個鬼腳圖呢?筆者採取的方法是：先把最左邊的答案完成，再依序往右完成每個答案。

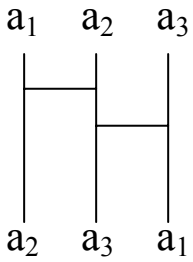
以下舉個例子：將 a_1、a_2、a_3、換成 a_2、a_3、a_1 則步驟如下：



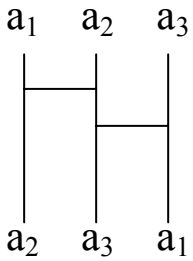
1.畫出將 a_2 換到定位，也就是第一條線的下方所需的橫線：



2.畫出將 a_3 換到定位，也就是第二條線的下方所需的橫線，且後加入的橫線其位階均需較先加入的橫線低：



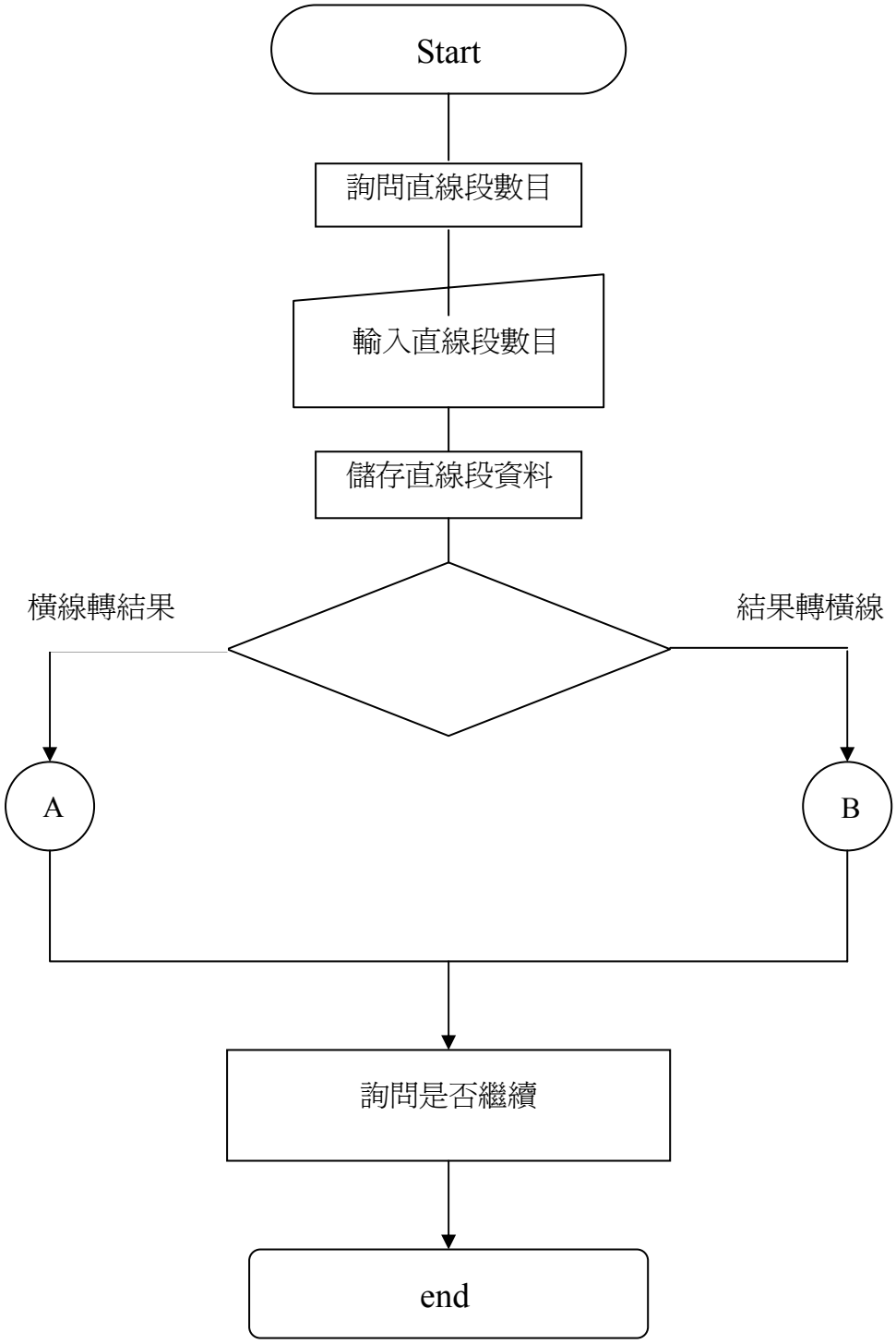
3.畫出將 a_1 換到定位，也就是第三條線的下方所需的橫線，由於原本的線條已經足以令 a_1 就定位，所以不需再畫橫線：



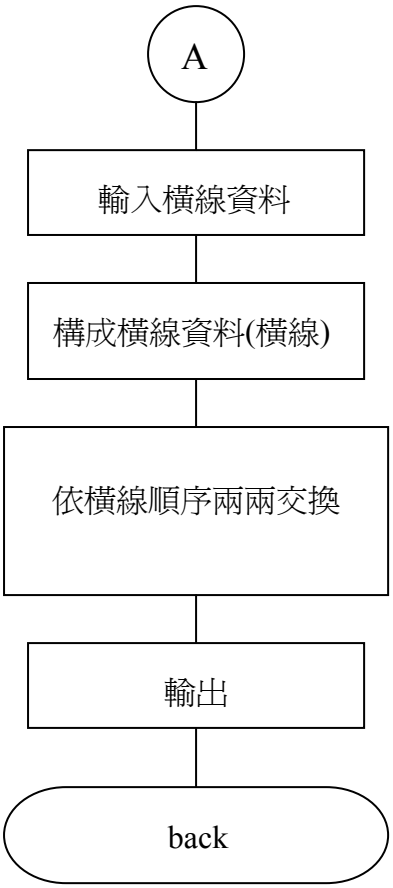
按照這個步驟，即可將 a_1、a_2、a_3、換成 a_2、a_3、a_1。

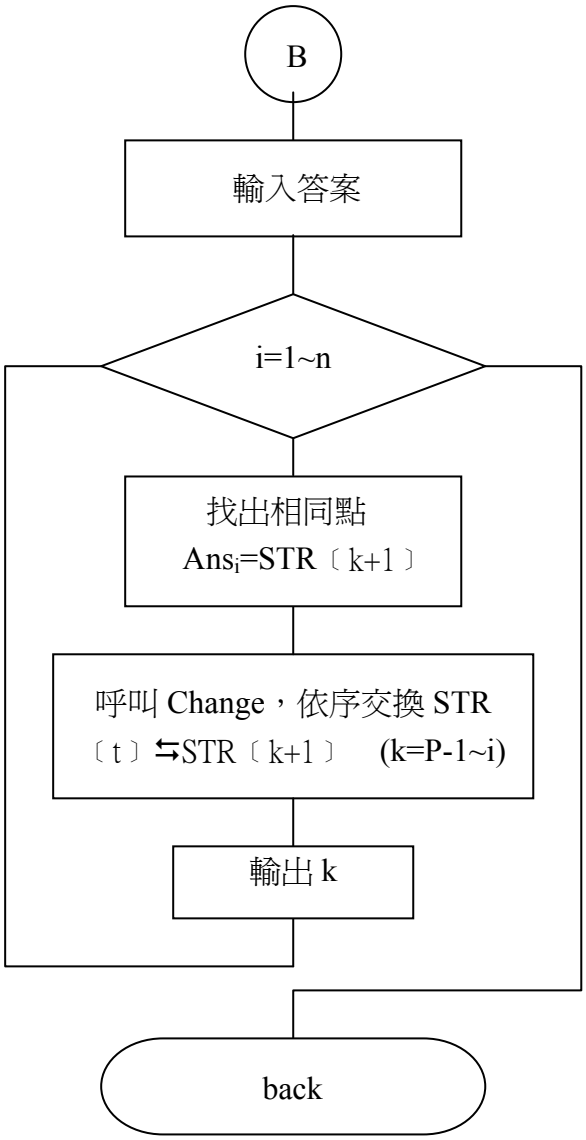
（四）程式流程圖

主程式：

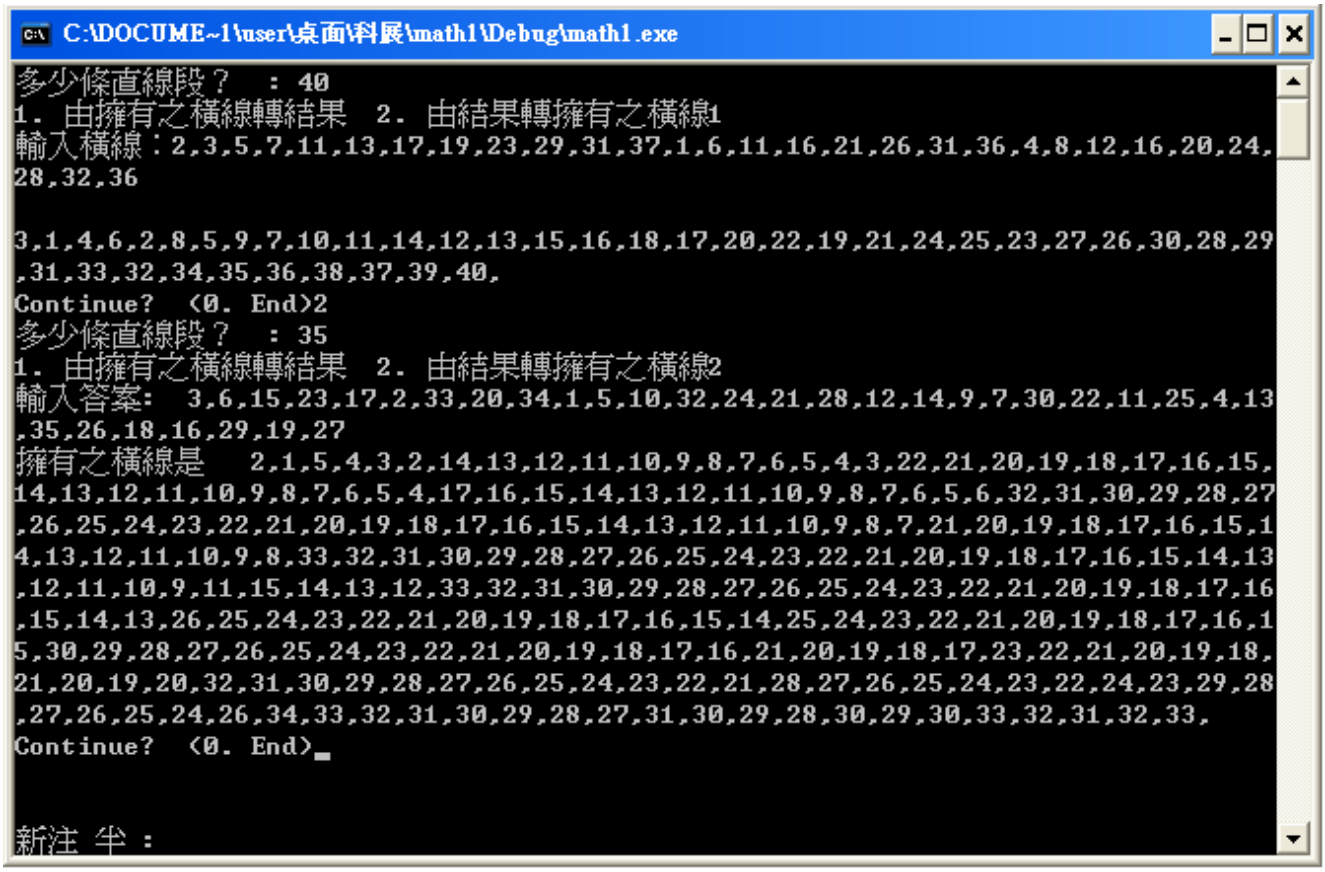


選擇 A 或 B 後的副程式:


A
輸入橫線資料
構成橫線資料(橫線)
依橫線順序兩兩交換
輸出
back


B
輸入答案
i=1~n
找出相同點
Ans$_i$=STR〔k+1〕
呼叫 Change，依序交換 STR
〔t〕⇆STR〔k+1〕 (k=P-1~i)
輸出 k
back

程式使用範例：



（五）鬼腳圖是否具一對一性：

鬼腳圖有沒有可能有兩個起點走到相同的終點呢？以下是我們的探討。

欲成立函數 f：$\{a_1,a_2,a_3,.....a_n\}\rightarrow\{b_1,b_2,b_3.....b_n\}$。

通過位階 1 的橫線 $1(L_k，L_{k+1})$後，成立函數 f_1：$\{a_1,a_2,a_3,.....a_n\}\rightarrow\{c_1,c_2,c_3.....c_n\}$，其中 $a_k=b_{k+1}$，$a_{k+1}=b_k$，其餘函數 $a_m=b_m$，且 $k,m\in\{x \mid 1\leqq x\leqq n-1，x\in N\}$，則此函數 f_1 為一對一交換。

如此一來,每通過一條橫線，就創造出一個一對一函數 f_k，而函數 f 即是所有 f_k 的合成函數，而一對一函數的合成當然是一對一函數，所以整個鬼腳圖代表的函數 f 亦為一對一函數，由此可知鬼腳圖必為一對一性。

（六）鬼腳圖的答案是否具不限定性

有沒有可能有某些起點，無論線怎麼畫，都不可能走到某些終點呢？以下是我們的探討。

欲成立函數 $f:\{a_1,a_2,a_3,\ldots\ldots..a_n\}\rightarrow\{b_1,b_2,b_3\ldots\ldots..b_n\}$，其中 $a_1=b_m$。

欲完成函數 $f_1:\{a_1,a_2,a_3,..,a_m...a_n\}\rightarrow\{a_m,a_2,a_3\ldots,a_1\ldots a_n\}$，則需作線條 $1(L_1，L_2)$、$2(L_2，L_3)$、……、$(m-2)(L_{m-2}，L_{m-1})$、$(m-1)(L_{m-1}，L_m)$、$m(L_{m-2}，L_{m-1})$、$(m+1)(L_{m-3}，L_{m-2})$……、$(2m-4)(L_1，L_2)$、$(2m-3)(L_1，L_2)$即可完成函數 f_1，使 a_1 置換為 b_1，a_m 置換為 b_m。

由此可知，若欲成立函數 $f_k:\{a_1,a_2,...a_k,..,a_h...a_n\}\rightarrow\{a_1,a_2,\ldots a_h\ldots,a_k\ldots a_n\}$，則作線條 $1(L_k，L_{k+1})$、$2(L_{k+1}，L_{k+2})$、……、$(h-k-1)(L_{h-2}，L_{h-1})$、$(h-k)(L_{h-1}，L_h)$、$(h-k+1)(L_{h-2}，L_{h-1})$、$(h-k+2)(L_{h-3}，L_{h-2})$……、$(2h-2k-2)(L_2，L_3)$、$(2h-2k-1)(L_1，L_2)$即可完成，上述步驟重複進行 n/2 次後即可完成函數 $f:\{a_1,a_2,a_3,\ldots\ldots..a_n\}\rightarrow\{b_1,b_2,b_3\ldots\ldots..b_n\}$。

由上述步驟，製造 f 即可將$\{a_1,a_2,a_3,\ldots\ldots..a_n\}$中的元素任意對調成為$\{b_1,b_2,b_3\ldots\ldots..b_n\}$，故鬼腳圖的答案具不限定性。

（七）鬼腳圖的畫法是否唯一

前面已證明鬼腳圖的函數 $f:\{a_1,a_2,a_3,...a_n\}\rightarrow\{b_1,b_2,b_3...b_n\}$是一對一函數，且可以將 $a_1,a_2,a_3,\ldots\ldots..a_n$ 任意重新排列成 $b_1,b_2,b_3\ldots\ldots..b_n$。可是同一個函數 f 中所使用的所有的橫線 x（$L_k$，$L_{k+1}$）是否只有一種排列方式?

如果有 $a_1,a_2,a_3,...a_n$ 共 n 個數據，則 $b_1,b_2,b_3...b_n$ 共有 n!種排列方式，也就是說，當有 n 條線時，鬼腳圖的答案共有 n!種。

不過，不管有幾條直線段，橫線的數目都可以無限制增加，所以橫線的畫法共有無限多種。

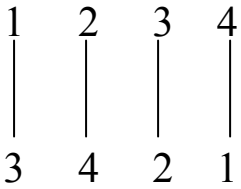
一個是有限，一個是無限，很顯然的，一種排列方式不可能只有一種橫線畫法，所以鬼腳圖的畫法不是唯一的。

五、討論：

（一）鬼腳圖的化簡

之前證明同一組結果鬼腳圖畫法不只一種，但是是否有些畫法是「濫竽充數」，也就是可以再化簡呢？

先以 4 條線為研究對象，現在將 a_1,a_2,a_3,a_4 換成 a_3,a_4,a_2,a_1（以下省略 a，僅用 n 表示各個 a，如 a_1 表示為 1）



（左圖橫線尚未完成）

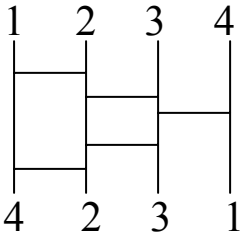
我們看到：1 移到 4 的位置，4 移到 2 的位置，2 移到 3 的位置，3 移到 1 的位置

我們將這個結果用群論中的「循環」表示成為：

$\begin{pmatrix}1,2,3,4\\4,3,1,2\end{pmatrix}$=(1,4,2,3)=(1,4)(1,2)(1,3)。

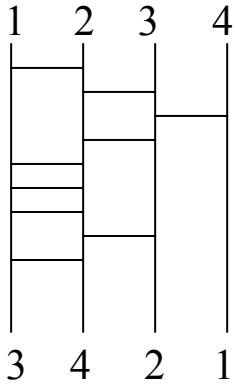
又我們須將所有的循環化為(k, k+1)形式的對換，因為鬼腳圖中的橫線均為(L_k，L_{k+1})的形式。

觀察(1,4)：(1,4)意為將 1 和 4 位置對調，所以需畫橫線 1(L_1，L_2)、2(L_2，L_3)、3 (L_3，L_4)、4 (L_2，L_3)、5(L_1，L_2)方可將 1，4 的位置對調：



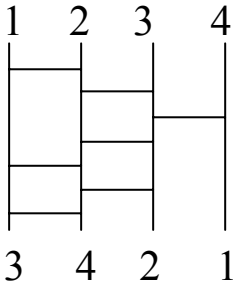
（以下將 L_k 省略為 k，且直接以書寫之先後順序表示位階 x）

由此可知，(1,4)可表示為(1,2)(2,3)(3,4)(2,3)(1,2)。同理，(1,3)可表示為(1,2)(2,3)(1,2)。於是我們可以令原循環(1,4,2,3)= (1,4)(1,2)(1,3) = (1,2)(2,3)(3,4)(2,3)(1,2)(1,2)(1,2)(2,3)(1,2)，作圖如下：



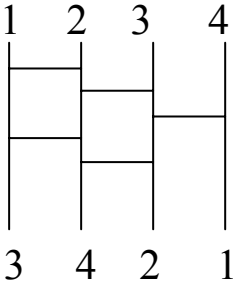
又我們可以發現：當有兩條線為(k,k+1)(k,k+1)時，原本走到 k 的數據會先被換到 k+1 再被換回 k，所以這兩條線根本是無用的，可以刪除。所以我們得到(k,k+1)(k,k+1)=(k,k)=不需畫線，以下稱這個結果為「成對抵消性」。

因此，原式=(1,2)(2,3)(3,4)(2,3)(1,2)(2,3)(1,2)，此鬼腳圖可簡化為：



即使已經簡化，仍然感覺稍嫌複雜，且隱約感覺仍可更進一步簡化。以下是我們的探討。

現在循環已改寫為(1,2)(2,3)(3,4)(2,3)(1,2)(2,3)(1,2)，其中(1,2)(2,3)(1,2)=(1,3)=(3,1)=(3,2)(1,2)(2,3)，這個結果亦可推廣為：(k,k+1)(k+1,k+2)…..(h-2,h-1)(h-1,h)(h-2,h-1).….(k+1,k+2)(k,k+1)=(k,h)=(h,k)=(h-1,h)(h-2,h-1)…..(k+1,k+2)(k,k+1)(k+1,k+2)….(h-2,h-1)(h-1,h)（其中 h>k，h、k∈N），以下稱這個結果為「交換性」。因此，可以將(1,2)(2,3)(3,4)(2,3)(1,2)(2,3)(1,2)再簡化為(1,2)(2,3)(3,4)(2,3)(2,3)(1,2)(2,3)=(1,2)(2,3)(3,4)(1,2)(2,3)，此鬼腳圖為：



並且我們發現：雖然我們明確規定按照位階排列應是(3,4)(1,2)，但是將其對調改為(1,2)(3,4)亦無不可，由此可以推論：(n,n+1)(m,m+1)=(m,m+1)(n,n+1)，其中 n≠m±1。更可以進一步得到結論：任意 x 條橫線，若沒有任兩條橫線 p(n,n+1)、q(m,m+1)出現 n=m±1 的情形，則這 x 條橫線的位階是可以任意對調的，以下稱這個結果為「位階不定性」。

（二）不同畫法的最簡鬼腳圖

由上述的成對抵消性和交換性可以將鬼腳圖化到最簡。但是由循環和交換性我們也可以知道：最簡鬼腳圖的畫法也不只一種。

由（一）中的例子：鬼腳圖可表示為循環$\begin{pmatrix}1,2,3,4\\4,3,1,2\end{pmatrix}$=(1,4,2,3)，但是，正如其名稱，循環的起點是可以任意取的，所以(1,4,2,3)=(4,2,3,1)=(2,3,1,4)=(3,1,4,2)。以下將它們都轉換為最簡鬼腳圖：

(1,4,2,3)= (1,2)(2,3)(3,4)(1,2)(2,3)　⇦這個前面做過了

(4,2,3,1)=(4,2)(4,3)(4,1)

=(3,4)(2,3)(3,4)(3,4)(3,4)(2,3)(1,2)(2,3)(3,4)

=(3,4)(2,3)(3,4)(2,3)(1,2)(2,3)(3,4)　⇦成對抵消性

=(2,3)(3,4)(2,3)(2,3)(1,2)(2,3)(3,4)　⇦交換性

=(2,3)(3,4)(1,2)(2,3)(3,4)*　　⇦成對抵消性

(2,3,1,4)=(2,3)(2,1)(2,4)

=(2,3)(1,2)(2,3)(3,4)(2,3)*

(3,1,4,2)=(3,1)(3,4)(3,2)

=(2,3)(1,2)(2,3)(3,4)(2,3)*

現在我們對這個鬼腳圖有 4 種畫法，再將每一種畫法用交換性改為其他的畫法（位階不定性只是簡化中可用的一種工具，改變位階後整個鬼腳圖仍然不變）：

(1,4,2,3)= (1,2)(2,3)(3,4)(1,2)(2,3)--1

=(1,2)(2,3)(1,2)(3,4)(2,3)

=(2,3)(1,2)(2,3)(3,4)(2,3)--2

=(2,3)(1,2)(3,4)(2,3)(3,4)--3

(4,2,3,1)= (2,3)(3,4)(1,2)(2,3)(3,4)=3

=(2,3)(1,2)(3,4)(2,3)(3,4)=3

=(2,3)(1,2)(2,3)(3,4)(2,3)=2

(2,3,1,4)=(2,3)(1,2)(2,3)(3,4)(2,3)=2

=(2,3)(1,2)(3,4)(2,3)(3,4)=3

(3,1,4,2)=(2,3)(1,2)(2,3)(3,4)(2,3)=2

=(2,3)(1,2)(3,4)(2,3)(3,4)=3

所以一共有 3 種畫法，而且無論是哪一種畫法，都可以由任何一個循環推導出來。所以，之後我們要找出鬼腳圖的畫法，只要統一由 1 起頭的循環推導出來就可以了。

六、結論：

1. 鬼腳圖是一種交換的概念，當我們在兩條線之間畫一條橫線時，其實就是把兩條線上原本的數據對調。
2. 鬼腳圖具一對一性：不可能有兩個起點走到相同的終點
3. 鬼腳圖的答案具不限定性：不管將原數據群怎樣重新排列，都可以畫出鬼腳圖。
4. 鬼腳圖的畫法不是唯一的：一種數據群重新排列的方式不可能只有一種畫橫線的方法。
5. 利用「成對抵消性」、「交換性」、「位階不限定性」可以將鬼腳圖化到十分簡潔。
6. 化簡後的鬼腳圖仍然不只一種，不過可以用「交換性」將其中一種化為所有的形式。其中只用「位階不限定性」改變後的形式並不算不同的形式

七、未來展望：

5. 研究環狀鬼腳圖
6. 研究空間鬼腳圖
7. 研究單向鬼腳圖
8. 研究不同方向鬼腳圖的疊加

附錄：程式碼:

```
// math1.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include "iostream.h"

Canswer_to_change(int,int*,int*);
Cchange_to_answer(int,int*);
Cchanging(int&,int&);


main()
{
    int n,choose,Icontinue=1;

    while(Icontinue != 0)
    {
        cout << "多少條直線段？   : ";
        cin >> n;

        int *start=new int[n];
        int *answer = new int [n];
```

```
        for(int i=0;i<n;i++)
        {
            start[i]=i+1;
            answer[i]=0;
        }

        cout << "1. 由擁有之橫線轉結果  2. 由結果轉擁有之橫線";
        cin >> choose;

        if(choose==1)
        {
            Cchange_to_answer(n,start);
        }

        else if(choose==2)
        {
            Canswer_to_change(n,start,answer);
        }

        cout << "Continue?   (0. End)";
        cin >> Icontinue;

        delete[] start;
        delete[] answer;
    }
}

Cchange_to_answer(int N,int* st)
{
    int n = N;
    int*start = st;
    int size=0,Clong=0;
    char change_enter[400];

    cout << "輸入橫線：";
    cin >> change_enter;

    while(change_enter[Clong] != '\0')
    {
        int a=0;
        while(change_enter[Clong] == ',' && a==0)
        {
            size++;
            a=1;
        }

        Clong++;
    }

    int*change = new int[size+1];
```

```
	for(int i=0;i<=size;i++)
	{
		change[i] = 0;
	}

	Clong=1;
	int h=0;

	change[0] = int( change_enter[0] ) - 48;

	while(change_enter[Clong-1] != '\0')
	{
		while( change_enter[Clong] != ',' && change_enter[Clong] != '\0')
		{
			change[h]= change[h]*10 + int( change_enter[Clong] ) - 48;

			Clong++;
		}

		h++;
		Clong++;
	}	//	存入橫線


	cout << "\n";

	for(i=0;i<=size;i++)
	{
		Cchanging(start[change[i]-1],start[change[i]]);
	}

	for( i = 0 ; i < n ; i++ )
	{
		cout << start[i] << ",";
	}
	cout << "\n";

	delete[] change;
}

Canswer_to_change(int N , int *st , int*ans)
{
	int n=N;
	int *start=st , *answer=ans;
	char answer_enter[400];

	int Along=1,g=0;

	cout << "輸入答案:  ";
```

```
	cin >> answer_enter;

	answer[0] = int( answer_enter[0] ) - 48;

	while(answer_enter[Along-1] != '\0')
	{
		while( answer_enter[Along] != ',' && answer_enter[Along] != '\0')
		{
			answer[g]= answer[g]*10 + int( answer_enter[Along] ) - 48;

			Along++;
		}

		g++;
		Along++;
	}

	int count=0;

	cout << "擁有之橫線是	";

	for(int i=0;i<n;i++)
	{
		int Looking;

		for(int j=i;j<n;j++)
		{
			if(answer[i] == start[j])
			{	Looking = j;	}	//	找出相同點

		}

		for(int k = Looking-1 ; k >= i ; k--)
		{
				Cchanging(start[k],start[k+1]);
				cout << k+1 << ",";
		}
	}

	cout << "\n";
}

Cchanging(int &change1,int &change2)
{
	int turn;

	turn = change1;
	change1 = change2;
	change2 = turn;
}
```