中華民國第 57 屆中小學科學展覽會

作品說明書

高級中等學校組　數學科

050416

棋盤上的飛舞機器人

學校名稱：新北市立中和高級中學

作者：	指導老師：
高二 高辰維	劉鴻儒
高二 凌中謙	王晞安
高二 魏家豪	

關鍵詞：圖論、矩陣、時間複雜度

摘要

本研究從圖論的角度探討桌遊「Micro Robots」所引出的相關問題。

第一部分，提出並證明同心圓法、鄰接矩陣元素的加法、鄰接矩陣的乘法，可以計算遊戲地圖中任兩點的最短距離、最短路徑與最短路徑數，分別將三個演算法撰寫 Python 程式，並比較其功能與時間複雜度。

第二部分，探討遊戲地圖上的參數：兩點平均距離的極值，並引出遊戲地圖設計的問題。

第三部分，探討鄰接矩陣、卡牌矩陣、位置矩陣與地圖的關係，進而提出一套演算法，以處理滿足給定條件地圖之存在性與設計方法，並將演算法撰寫 Python 程式。

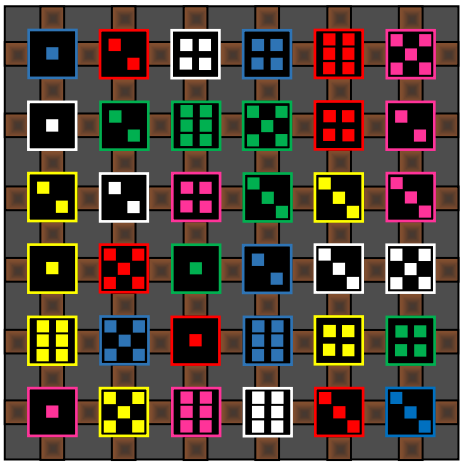
壹、遊戲介紹與研究定位

一、Micro Robots 遊戲介紹

【放置】

由 6×6 共 36 個均相異格子(如圖 1)所組成的地圖(如圖 2)，每個格子是由一個顏色與一個數字組成，其中顏色有 6 種，數字為 1 到 6。

（圖 1-黃色 1 點）



（圖 2-地圖）

【規則】

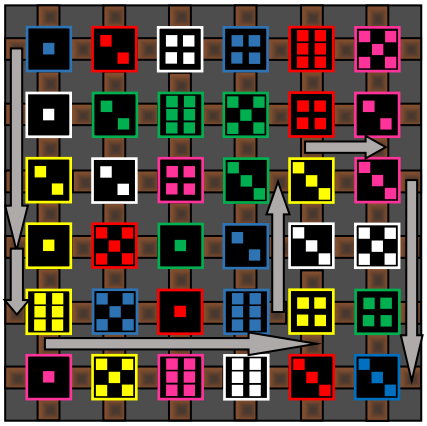
玩家任選一個格子做為起點，再任選起點以外的格子做為終點，試著遵循以下的規則由起點移動到終點：

玩家由目前的所在格，搜尋所在格同一行或同一列的格子，若與目前的所在格有相同顏色或數字，則可以移動過去，並稱這個過程為移動 1 步。

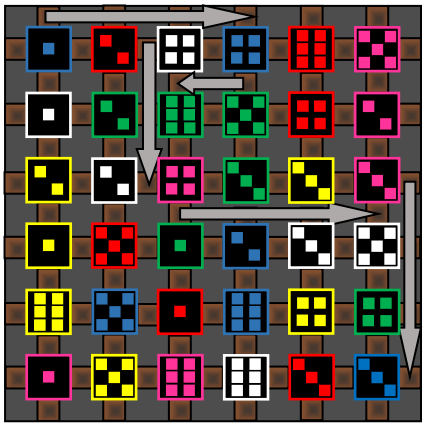
移動過去後，重複上述過程，直到移動至終點。若能以最少的步數走到終點即為遊戲贏家，而這個最少步數稱為這兩個格子之間的距離。

【範例】

給定地圖(圖 2)，若以  為起始點，  為終點，我們有多種的移動方法，舉例如下：



(移動6步)



(移動5步)

（圖 3）

右邊走法只移動5步，而左邊走法為6步，故移動較少的右邊玩家勝利。

二、研究定位

我們以 Google、Google scholar 搜尋，並參閱台灣國際科展與全國科展資料庫，發現沒有相關研究探討 Micro Robots 這款桌遊。而歷屆科展中的棋盤問題都是在處理：「如何在固定的 Graph 上找 Hamiltonian path 、cycle (例如騎士問題)」，而本研究的「遊戲地圖」與 Graph 是可變動的，我們除了處理最短路徑問題外，也研究了在給定 Graph 下反找「遊戲地圖」的問題。

本研究的第三部分很容易跟歷屆作品混淆，我們的研究目的不是要找 Hamiltonian path、cycle，而是要設計一個「遊戲地圖」，使其 Graph 只能是 Hamiltonian path。我們提供的演算法，適用任何的 Graph，不僅僅侷限在 Hamiltonian path，只要有 Graph，都可以判別「遊戲地圖」的存在性。

貳、研究目的

關於本研究，我們主要探討給定「遊戲地圖」下的找解，以及設計「遊戲地圖」：

一、提出三種方法找任兩點的最短距離、最短路徑與最短路徑數，並給出證明與寫成 Python 程式。

二、探討地圖任兩點平均距離的極值。

三、提出滿足特殊條件的地圖之設計方法，並寫成 Python 程式，最後將其應用在地圖的極端情形(平均距離的極值)。

參、研究工具

紙、筆、Microsoft Office Word、Microsoft Office Excel、Python

肆、名詞解釋

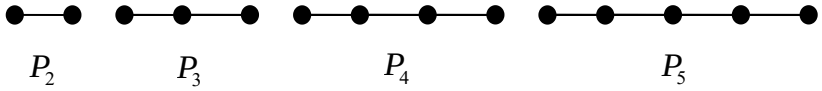
一、Graph $G=G(V,E)$

Graph G 是一個有序對 $G=G(V,E)$，其中 V 是非空的有限集，其元素稱為頂點，E 是一些 V 的相異二元序對構成的集合，即 $E\subseteq\{e\mid e\subseteq V,|e|=2\}$，其元素稱為邊。

★ $V(G)$ 表示 G 的點集，$E(G)$ 表示 G 的邊集。

★ 邊 $d=\{u,v\}$ 直接寫成 uv，此時稱 u 跟 v 是 d 的端點、d 和 u (及 v)相連、u 和 v 相鄰。

二、在圖論中，若有 n 個點排成 $v_1, v_2, ..., v_n$ 的序列，並且滿足所有邊 e_k 的構成只有連接 v_k 與 v_{k+1} $\forall k = 1, 2, ..., n-1$，即 $v_1, e_1, v_2, e_2 \cdots, e_{n-1}, v_n$，則我們稱此圖為 P_n Graph，舉例如下：



（圖 4）

三、路徑是一條點邊相間的序列 $v_1, e_1, v_2, e_2 \cdots, e_{n-1}, v_n$，其中 v_1 稱為路徑的起點，v_n 稱為路徑的終點，$n-1$ 為路徑的長。

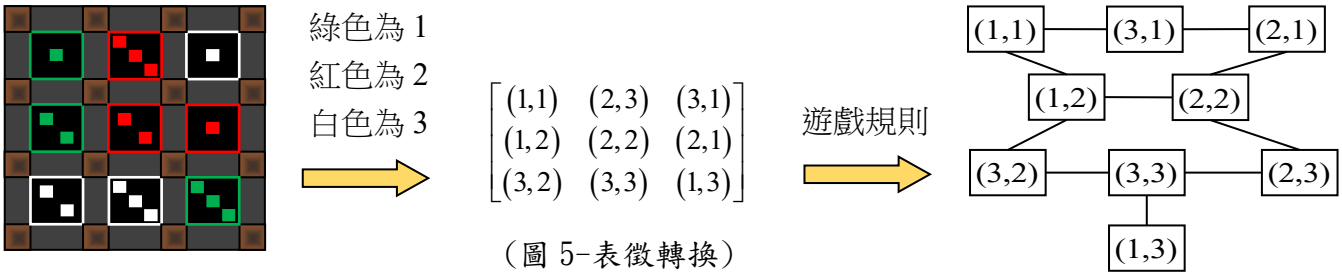
★ 連接兩頂點 v_i 、 v_j 的所有路徑長之最小值稱為 v_i 、 v_j 的距離，記為 $\rho(v_i, v_j)$。

★ 若 Graph G 中任兩點之間都有路徑連接，則 G 稱為連通圖。

★ 若 Graph G 為連通圖，則 $\max_{u,v \in V(G)} \rho(u,v)$ 稱為 G 的直徑，記為 $\text{diam}(G)$。

四、地圖(Map)與圖(Graph)的轉換

為了推廣與撰寫程式的便利性，將原本的格子中的顏色及數字表示成(顏色，數字)，若令綠色為1、紅色為2、白色為 3，則可將「遊戲地圖」以數對矩陣表示，往後皆以數字表徵處理，再根據遊戲規則，將數對矩陣轉換成 Graph G，如圖 5 所示：



（圖 5-表徵轉換）

五、地圖的矩陣表示法

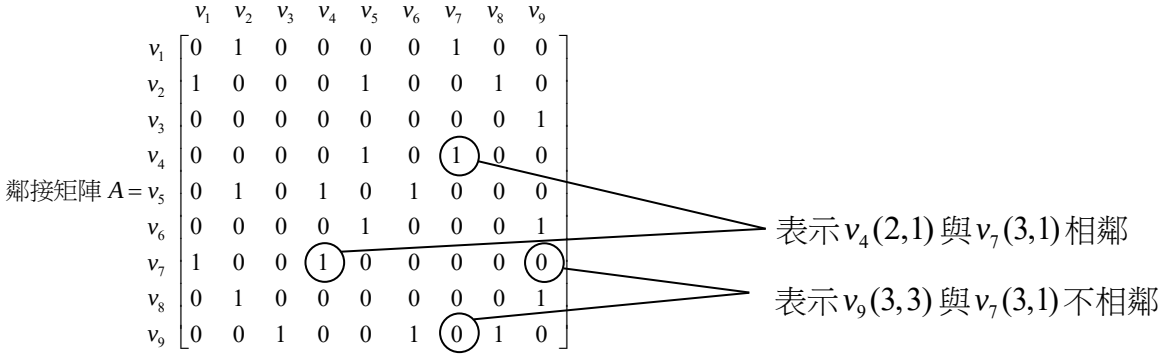
將 k 種顏色，k 個數字的地圖轉換成 Graph G，即 $V(G) = \{(p,q) : p = 1, 2, \cdots k, q = 1, 2, \cdots k\}$，並將頂點依序標籤如下：$(p,q) = v_{k(p-1)+q}$。

以圖 5 為例，$3^2 = 9$ 個點的編號為：

$$(1,1) = v_1 \ , \ (1,2) = v_2 \ , \ (1,3) = v_3 \ ,$$
$$(2,1) = v_4 \ , \ (2,2) = v_5 \ , \ (2,3) = v_6 \ ,$$
$$(3,1) = v_7 \ , \ (3,2) = v_8 \ , \ (3,3) = v_9 \ ,$$

底下的 $k^2 \times k^2$ 矩陣的 (i, j) 元皆記錄頂點 v_i 與 v_j 的關係，並以圖 5 為例：

★ 鄰接矩陣 A：若 v_i 與 v_j 相鄰，則 $A_{ij} = 1$，否則 $A_{ij} = 0$



★ 方法數矩陣 N ： $N_{ij}=$ 連接v_i與v_j的最短路徑數

$$N=\begin{bmatrix}0&1&1&1&1&1&1&1&1\\1&0&1&1&1&1&1&1&1\\1&1&0&1&1&1&\textcircled{2}&1&1\\1&1&1&0&1&1&1&1&1\\1&1&1&1&0&1&1&1&1\\1&1&1&1&1&0&1&1&1\\1&1&\textcircled{2}&1&1&1&0&\textcircled{1}&2\\1&1&1&1&1&1&\textcircled{1}&0&1\\1&1&1&1&1&1&2&1&0\end{bmatrix}$$

表示$v_3(1,3)$與$v_7(3,1)$有 2 種最短路徑P_3

表示$v_7(3,1)$與$v_8(3,2)$有 1 種最短路徑

★ 距離矩陣S ： $S_{ij}=\rho(v_i,v_j)$

$$S=\begin{bmatrix}0&1&4&\textcircled{2}&2&3&1&2&3\\1&0&3&2&1&2&2&1&2\\4&3&0&4&3&2&5&2&1\\\textcircled{2}&2&4&0&1&2&1&3&3\\2&1&3&1&0&2&2&2&2\\3&2&2&2&1&0&3&2&\textcircled{1}\\1&2&5&1&2&3&0&3&4\\2&1&2&3&2&2&3&0&1\\3&2&1&3&2&\textcircled{1}&4&1&0\end{bmatrix}$$

表示$v_1(1,1)$與$v_4(2,1)$最少移動 2 步可到(距離為 2)

表示$v_6(2,3)$與$v_9(3,3)$最少移動 1 步可到(距離為 1)

★ 卡牌矩陣C ：若v_i 、 v_j的第 1 個元相同或第 2 個元相同，則$C_{ij}=1$ ，否則$C_{ij}=0$

$$C=\begin{bmatrix}1&1&1&\textcircled{1}&0&0&1&0&0\\1&1&1&0&1&0&0&1&0\\1&1&1&0&0&1&0&0&1\\\textcircled{1}&0&0&1&1&1&1&0&0\\0&1&0&1&1&1&0&1&0\\0&0&1&1&1&1&0&\textcircled{0}&1\\1&0&0&1&0&0&1&1&1\\0&1&0&0&1&\textcircled{0}&1&1&1\\0&0&1&0&0&1&1&1&1\end{bmatrix}$$

表示$v_1(1,1)$與$v_4(2,1)$有共同的元，數字相同

表示$v_6(2,3)$與$v_8(3,2)$共同的元數字皆不同

★ 位置矩陣P ：若v_i 、 v_j的「遊戲地圖」位置在同一行或同一列，則$P_{ij}=1$ ，否則$P_{ij}=0$

$$P=\begin{bmatrix}1&1&\textcircled{0}&0&0&1&1&1&0\\1&1&0&1&1&0&0&1&0\\\textcircled{0}&0&1&1&0&0&1&1&1\\0&1&1&1&1&0&1&0&0\\0&1&0&1&1&1&0&0&1\\1&0&0&0&1&1&\textcircled{1}&0&1\\1&0&1&1&0&\textcircled{1}&1&0&0\\1&1&1&0&0&0&0&1&1\\0&0&1&0&1&1&0&1&1\end{bmatrix}$$

表示$v_1(1,1)$與$v_3(1,3)$在地圖的不同行且不同列

表示$v_6(2,3)$與$v_7(3,1)$在地圖的同行或同列

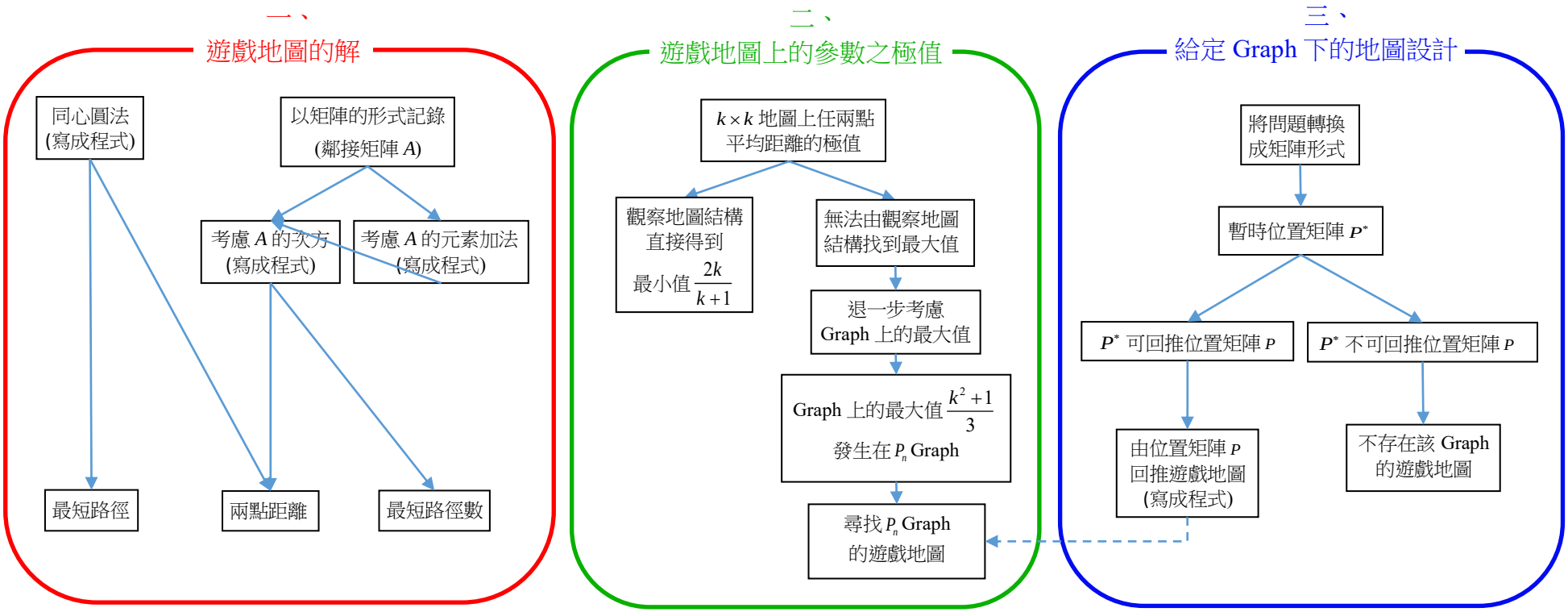
★ 暫時位置矩陣P^* ：當$i=j$ ， $P^*_{ij}=1$ 。

當$i\neq j$ ，若$C_{ij}\neq 0$ ，則$P^*_{ij}=\dfrac{A_{ij}}{C_{ij}}$ ，否則$P^*_{ij}=_$ (待定數值)

$$P^*=\begin{bmatrix}1&1&0&0&\_&\_&1&\_&\_\\1&1&0&\_&1&\_&\_&1&\_\\0&0&1&\_&\_&0&\_&\_&1\\0&\_&\_&1&1&0&1&\_&\_\\\_&1&\_&1&1&1&\_&0&\_\\\_&\_&0&0&1&1&\_&\_&1\\1&\_&\_&1&\_&\_&1&0&0\\\_&1&\_&\_&0&\_&0&1&1\\\_&\_&1&\_&\_&1&0&1&1\end{bmatrix}$$

伍、研究架構圖

考慮 $k \times k$ 的遊戲地圖



結合上述研究結果，

在給定遊戲地圖下，我們可以透過(研究一)找到解(最短路徑、兩點距離、最短路徑數)，

若希望設計滿足特殊條件的地圖(研究二)，我們可以先考慮 Graph 的設計，

進而透過(研究三)找到滿足的遊戲地圖。

陸、研究結果

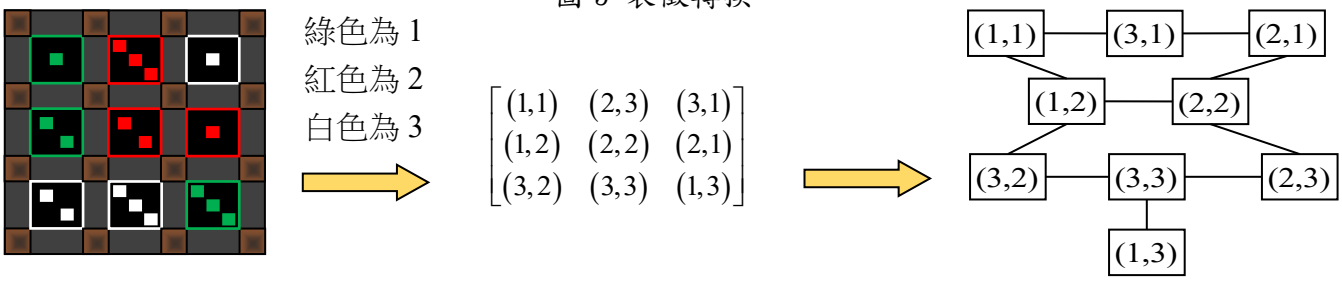
本研究主要分成三部分，第一部分，探討給定「遊戲地圖」下，任兩點的最短距離、最短路徑與最短路徑數。第二部分，接續兩點的最短距離，我們從「遊戲地圖」中兩點平均距離的角度出發，討論了平均距離的極值。而研究的過程中我們發現，若要設計遊戲地圖以滿足平均距離有極值，我們可以先考慮 Graph 上的極值，再設法由 Graph 找到「遊戲地圖」，因此我們著手進行第三部分，提出給定 Graph 下，「遊戲地圖」的設計方法，並將其應用在地圖的極端情形(平均距離的極值)上。

一、提出三種方法找任兩點的最短距離、最短路徑與最短路徑數，並給出證明與寫成 Python 程式

為了方便說明，我們皆以圖 5 為例說明，並提出一般化的作法。

(1) 同心圓法

圖 5-表徵轉換



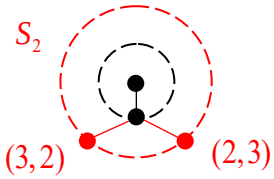
將「遊戲地圖」轉換成 Graph，若以 $(1,3)$ 為起始點，我們計算它到其它各點距離如下：

Step1 我們以 $(1,3)$ 為圓心，令 $S_0=\{(1,3)\}$。

Step2 收集與 S_0 的元素距離為 1 的點，可得 $(3,3)$，令 $S_1=\{(3,3)\}$，則 S_1 的元素與 $(1,3)$ 距離為 1。

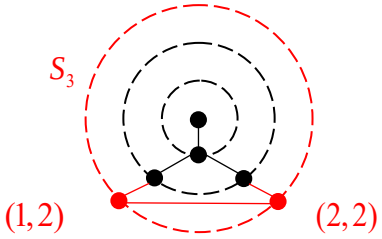
Step3 扣除 $S_0 \cup S_1$ 後剩下的點中，收集與 S_1 的元素距離為 1 的點，可得 $(3,2),(2,3)$，

令 $S_2=\{(3,2),(2,3)\}$，顯然 S_2 的元素與起始點 $(1,3)$ 的距離皆為 2。



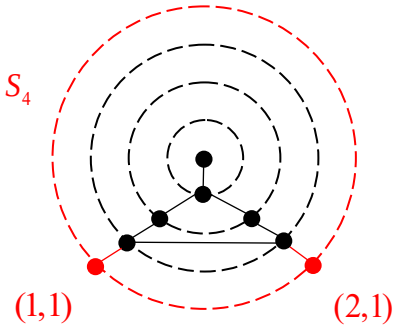
Step4 扣除 $S_0 \cup S_1 \cup S_2$ 後剩下的點中，收集與 S_2 的元素距離為 1 的點，可得 $(1,2),(2,2)$，

令 $S_3=\{(1,2),(2,2)\}$，顯然 S_3 的元素與起始點 $(1,3)$ 的距離皆為 3。



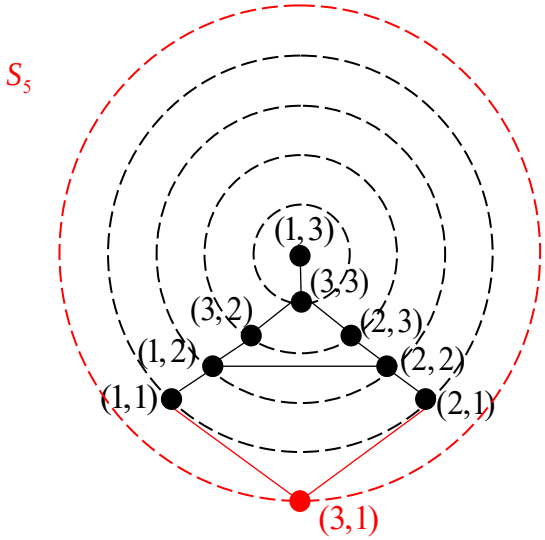
Step5 扣除 $\bigcup_{k=0}^{3} S_k$ 後剩下的點中，收集與 S_3 的元素距離為1的點，可得 $(1,1),(2,1)$ ，

令 $S_4=\{(1,1),(2,1)\}$ ，顯然 S_4 的元素與起始點(1,3)的距離皆為4 。



Step6 扣除 $\bigcup_{k=0}^{4} S_k$ 後剩下的點中，收集與 S_4 的元素距離為1的點，可得(3,1) ，

令 $S_5=\{(3,1)\}$ ，顯然 S_5 的元素與起始點(1,3)的距離皆為5 。



我們若將 Graph 畫在同心圓上(如上圖)，我們可以從中得知 $(1,3)$ 與任意一點的距離、最短路徑的走法。例如： $(1,3)$ 與 $(3,1)$ 的距離為5 ，而其路徑有兩種，分別為

$$(1,3)\to(3,3)\to(3,2)\to(1,2)\to(1,1)\to(3,1)$$

$$(1,3)\to(3,3)\to(2,3)\to(2,2)\to(2,1)\to(3,1)$$

一般化作法如下：

我們由 $S_0=\{u_0\}$ 開始，分階段找出一串集合 $S_0,S_1,S_2,\ldots$ ，第 i 個階段完成後會得到由 u_0 到 S_i 各點的距離與最短路徑。

第一階段是由 $S_0=\{u_0\}$ 開始，計算 $\rho\left(u_0,S_0'\right)$ (其中 $S_0'=V(G)\setminus S_0$)，從而找出 $u_{11},u_{12},\cdots,u_{1n_1}\in S_0'$ 使得 $\rho(u_0,u_{1k})=\rho\left(u_0,S_0'\right)=1,\forall k=1,2\cdots,n_1$ ，並令 $\{u_{11},u_{12},\cdots,u_{1n_1}\}=S_1$ 。

第 i 階段，計算 $\rho(S_{i-1},(\bigcup_{k=0}^{i-1}S_k)')$ ，從而找出 $u_{i1},u_{i2},\cdots,u_{in_i}\in(\bigcup_{k=0}^{i-1}S_k)'$ 使得

$\rho(u_0,u_{ik})=\rho\left(u_0,(\bigcup_{k=0}^{i-1}S_k)'\right)=i,\forall k=1,2\cdots,n_i$ ，並令 $\{u_{i1},u_{i2},\cdots,u_{in_i}\}=S_i$ 。

顯然，若 $G(V,E)$ 為連通圖，只要 m 足夠大，我們有

$\bigcup_{k=0}^{m}S_k=V$ ，這表示當 $v\in V$ ， $v\in S_i$ for some i ，且 $\rho(u_0,v)=i$

事實上，只要滿足 $\bigcup_{k=0}^{m}S_k=V$ ，則 Graph 為連通圖。

我們將上述過程撰寫 Python 程式，詳見(附錄一)。

定理 1：給定 Graph 上之頂點 u_0，依照同心圓法的流程建構集合序列 $\{S_i\}$，則當 $p\in S_n$，$\rho(p,u_0)=n$ 。

證明：當 $n=1$ 時，根據 S_1 的定義，顯然成立。

假設 $n\le k$ 時，命題皆成立，即當 $p\in S_n$ ， $\rho(p,u_0)=n$ ， $\forall n\le k$ 。

當 $n=k+1$ 時，根據 S_{k+1} 的定義，存在 $u_k\in S_k$ 使得 $\{u_k,p\}\in E(G)$，再由歸納假設，存在 $u_i\in S_i$ 使得 $\{u_i,u_{i+1}\}\in E(G)$ ， $\forall i\le k-1$，因此可以找到路徑 $u_0,u_1,.....,u_k,p$ ，故 $\rho(u_0,p)\le k+1$ 。

另一方面，假設 $\rho(u_0,p)=l<k+1$，則存在一路徑 $u_0,v_1,v_2,.....,v_{l-1},p$ ，根據 $\{S_i\}$ 的定義，顯然 $v_1\in S_1$ ， $v_2\in S_2$，因為是最短路徑，所以 $v_3\notin S_0\cup S_1\cup S_2$ ，故 $v_3\in S_3$，同理 $v_j\in S_j$ $\forall j$ ，這也表示 $p\in S_l$，其中 $l<k+1$，這顯然與 $p\in S_{k+1}\subset G\setminus\bigcup_{i=0}^{k}S_i$ 矛盾，故假設 $\rho(u_0,p)=l<k+1$ 不成立，因此 $\rho(u_0,p)=k+1$，故由數學歸納法得證。 ■

同心圓法是在 Graph 上，以一個頂點為圓心，一層層的搜尋，也就是說我們一次只能計算一個點(圓心)與其他點的距離，而我們希望能更有系統地計算出任兩點的距離。

在研究此問題的過程中，任何一個 $k\times k$ 的「遊戲地圖」，都可以轉換為有 $n=k^2$ 個頂點的 Graph，而每個 Graph 都可以用矩陣的形式記錄頂點相鄰的關係(名詞解釋中的鄰接矩陣)，我們希望藉由鄰接矩陣的運算，找出任兩點的距離，並進一步地求出最短路徑數。

(2) 鄰接矩陣的元素加法

鄰接矩陣 $A_{n\times n}$ 的定義如下：若 v_i 與 v_j 相鄰，則 $A_{ij}=1$，否則 $A_{ij}=0$ 。我們發現：

對於鄰接矩陣的非零元做加法可以有系統地計算出任兩點的距離。

我們希望透過若干步驟，找到答案，過程中以藍色的 0 表示不確定的距離。

我們以圖 5 為例：

Step1 給定鄰接矩陣 A ，令 $A_1 = A$

$$A_1 = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

若 $i \neq j$ ，則 $(A_1)_{ij} = 1$ 表示 v_i, v_j 的距離已經確定，而 $(A_1)_{ij} = 0$ 表示，距離不確定。

Step2 考慮 A_1 不確定的位置(0)，若存在 p 使得 $(A_1)_{ip} + (A_1)_{pj} = 2$ ，則 $(A_2)_{ij} = 2$ ，

否則維持是 0。即 $(A_2)_{ij} = 2$ 表示存在 p 使得 $\rho(v_i, v_p) + \rho(v_p, v_j) = 2$ ，

因為 $\rho(v_i, v_j) \leq \rho(v_i, v_p) + \rho(v_p, v_j) = 2$ ，又 $\rho(v_i, v_j)$ 在 A_1 中為不確定($\because \rho(v_i, v_j) \neq 1$)，

所以 $\rho(v_i, v_j) = 2$ 。

例如：因為 $(A_1)_{17} + (A_1)_{74} = 2$ ，故 $(A_2)_{14} = 2$ ；因為 $(A_1)_{89} + (A_1)_{93} = 2$ ，故 $(A_2)_{83} = 2$ 。

而 $(A_2)_{13} = 0$ 是因為 $(A_1)_{1k} + (A_1)_{k3}$ 恆不為 2 ，故 $\rho(v_1, v_3) \neq 2$ ， $\rho(v_1, v_3)$ 仍不確定。

$$A_1 = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & \textcircled{1} & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & \textcircled{1} & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \triangle 1 \\ 0 & 0 & \triangle 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \longrightarrow A_2 = \begin{bmatrix} 0 & 1 & 0 & \textcircled{2} & 2 & 0 & 1 & 2 & 0 \\ 1 & 0 & 0 & 2 & 1 & 2 & 2 & 1 & 2 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 & 2 & 1 \\ 2 & 2 & 0 & 0 & 1 & 2 & 1 & 0 & 0 \\ 2 & 1 & 0 & 1 & 0 & 1 & 2 & 2 & 2 \\ 0 & 2 & 2 & 2 & 1 & 0 & 0 & 2 & 1 \\ 1 & 2 & 0 & 1 & 2 & 0 & 0 & 0 & 0 \\ 2 & 1 & \triangle 2 & 0 & 2 & 2 & 0 & 0 & 1 \\ 0 & 2 & 1 & 0 & 2 & 1 & 0 & 1 & 0 \end{bmatrix}$$

簡單來說，要確定 $(A_2)_{ij}$ 只要去找 $(A_1)_{ik} + (A_1)_{kj}$ 的所有組合，只要有一個 k ，

讓其和為 2 ， $(A_2)_{ij}$ 就等於 2 。反過來說，如果沒有這樣的 k ，則 $(A_2)_{ij}$ 就維持是 0 ，

表示 $\rho(v_i, v_j)$ 仍為不確定。

Step3 考慮 A_2 中距離不確定的位置(0)，若存在 p 使得 $(A_2)_{ip} + (A_2)_{pj} = 3$ ，則 $(A_3)_{ij} = 3$ ，

否則維持是 0。即 $(A_3)_{ij} = 3$ 表示存在 p 使得 $\rho(v_i, v_p) + \rho(v_p, v_j) = 3$ ，

因為 $\rho(v_i, v_j) \leq \rho(v_i, v_p) + \rho(v_p, v_j) = 3$ ，又 $\rho(v_i, v_j)$ 在 A_2 中為不確定($\because \rho(v_i, v_j) \neq 1, 2$)

所以 $\rho(v_i, v_j) = 3$ 。

例如：因為 $(A_2)_{64} + (A_2)_{47} = 3$ ，故 $(A_3)_{67} = 3$ ；因為 $(A_2)_{59} + (A_2)_{93} = 3$ ，故 $(A_3)_{53} = 3$ 。

而 $(A_3)_{34} = 0$ 是因為 $(A_2)_{3k} + (A_2)_{k4}$ 恆不為 3 。

$$A_2=\begin{bmatrix}0&1&0&2&2&0&1&2&0\\1&0&0&2&1&2&2&1&2\\0&0&0&0&0&2&0&2&1\\2&2&0&0&1&2&1&0&0\\2&1&0&1&0&1&2&2&2\\0&2&2&2&1&0&0&2&1\\1&2&0&1&2&0&0&0&0\\2&1&2&0&2&2&0&0&1\\0&2&1&0&2&1&0&1&0\end{bmatrix}\longrightarrow A_3=\begin{bmatrix}0&1&0&2&2&3&1&2&3\\1&0&3&2&1&2&2&1&2\\0&3&0&0&3&2&0&2&1\\2&2&0&0&1&2&1&3&3\\2&1&3&1&0&1&2&2&2\\3&2&2&2&1&0&3&2&1\\1&2&0&1&2&3&0&3&0\\2&1&2&3&2&2&3&0&1\\3&2&1&3&2&1&0&1&0\end{bmatrix}$$

Step4 考慮 A_3 中距離不確定的位置(0)，若存在 p 使得 $(A_3)_{ip}+(A_3)_{pj}=4$，則 $(A_4)_{ij}=4$，否則維持是 0。即 $(A_4)_{ij}=4$ 表示存在 p 使得 $\rho(v_i,v_p)+\rho(v_p,v_j)=4$，

因為 $\rho(v_i,v_j)\le\rho(v_i,v_p)+\rho(v_p,v_j)=4$，又 $\rho(v_i,v_j)$ 在 A_3 中為不確定($\because \rho(v_i,v_j)\neq 1,2,3$)，所以 $\rho(v_i,v_j)=4$。

例如：因為 $(A_3)_{45}+(A_3)_{53}=4$，故 $(A_4)_{43}=4$；因為 $(A_3)_{96}+(A_3)_{67}=4$，故 $(A_4)_{97}=4$。

而 $(A_4)_{37}=0$ 是因為 $(A_3)_{3k}+(A_3)_{k7}$ 恆不為 4。

$$A_3=\begin{bmatrix}0&1&0&2&2&3&1&2&3\\1&0&3&2&1&2&2&1&2\\0&3&0&0&3&2&0&2&1\\2&2&0&0&1&2&1&3&3\\2&1&3&1&0&1&2&2&2\\3&2&2&2&1&0&3&2&1\\1&2&0&1&2&3&0&3&0\\2&1&2&3&2&2&3&0&1\\3&2&1&3&2&1&0&1&0\end{bmatrix}\longrightarrow A_4=\begin{bmatrix}0&1&4&2&2&3&1&2&3\\1&0&3&2&1&2&2&1&2\\4&3&0&4&3&2&0&2&1\\2&2&4&0&1&2&1&3&3\\2&1&3&1&0&1&2&2&2\\3&2&2&2&1&0&3&2&1\\1&2&0&1&2&3&0&3&4\\2&1&2&3&2&2&3&0&1\\3&2&1&3&2&1&4&1&0\end{bmatrix}$$

Step5 考慮 A_4 中距離目前不確定的位置(0)，若存在 p 使得 $(A_4)_{ip}+(A_4)_{pj}=5$，則 $(A_5)_{ij}=5$，否則維持是 0。即 $(A_5)_{ij}=5$ 表示存在 p 使得 $\rho(v_i,v_p)+\rho(v_p,v_j)=5$，

因為 $\rho(v_i,v_j)\le\rho(v_i,v_p)+\rho(v_p,v_j)=5$，又 $\rho(v_i,v_j)$ 在 A_4 中為不確定($\because \rho(v_i,v_j)\neq 1,2,3,4$)，所以 $\rho(v_i,v_j)=5$。

例如：因為 $(A_4)_{72}+(A_4)_{23}=5$，故 $(A_5)_{73}=5$；因為 $(A_4)_{74}+(A_4)_{43}=5$，故 $(A_5)_{73}=5$。

而除了對角線外沒有任何 $(A_5)_{ij}=0$，所以 A_5 距離矩陣 S。

$$A_4=\begin{bmatrix}0&1&4&2&2&3&1&2&3\\1&0&3&2&1&2&2&1&2\\4&3&0&4&3&2&0&2&1\\2&2&4&0&1&2&1&3&3\\2&1&3&1&0&1&2&2&2\\3&2&2&2&1&0&3&2&1\\1&2&0&1&2&3&0&3&4\\2&1&2&3&2&2&3&0&1\\3&2&1&3&2&1&4&1&0\end{bmatrix}\longrightarrow A_5=\begin{bmatrix}0&1&4&2&2&3&1&2&3\\1&0&3&2&1&2&2&1&2\\4&3&0&4&3&2&5&2&1\\2&2&4&0&1&2&1&3&3\\2&1&3&1&0&1&2&2&2\\3&2&2&2&1&0&3&2&1\\1&2&5&1&2&3&0&3&4\\2&1&2&3&2&2&3&0&1\\3&2&1&3&2&1&4&1&0\end{bmatrix}$$

一般化作法如下：

給定 Graph 的鄰接矩陣 A ，定義矩陣數列 $\{A_k\}$ 如下：

$A_1 = A$

$$(A_{k+1})_{ij} = \begin{cases} 0 & if \quad i = j \\ (A_k)_{ij} & if \quad 0 < (A_k)_{ij} < k+1 \ \wedge i \neq j \\ k+1 & if \quad (A_k)_{ij} = 0 \ \wedge \ \exists\, p \ni : (A_k)_{ip} + (A_k)_{pj} = k+1 \ \wedge \ i \neq j \\ 0 & if \quad \nexists\, p \ni : (A_k)_{ip} + (A_k)_{pj} = k+1 \wedge i \neq j \end{cases} \qquad \forall k \geq 1$$

定理 2：若 Graph 為連通圖且 A_m 滿足除了主對角線外其餘不為 0 ，則 A_m 即為距離矩陣 S 。

證明：當 $n=1$ 時，由於 $A_1 = A$ ，根據 A 的定義，顯然成立。

假設 $n \leq k$ 時，命題皆成立，即當 $0 < (A_k)_{ij} \leq k$ ， $\rho(v_i, v_j) = (A_k)_{ij}$ ， $\forall n \leq k$ 。

當 $n = k+1$ 時，根據 A_{k+1} 的定義，若 $(A_k)_{ij} \neq 0$ ，則 $(A_{k+1})_{ij} = (A_k)_{ij}$ ，由歸納假設 $(A_{k+1})_{ij} = (A_k)_{ij} = \rho(v_i, v_j)$ ，若 $(A_k)_{ij} = 0$ 且存在 p 使得 $(A_k)_{ip} + (A_k)_{pj} = k+1$ ，因為 $\rho(v_i, v_j) \leq \rho(v_i, v_p) + \rho(v_p, v_j) = k+1$ 且 $(A_k)_{ij} = 0$ ，根據歸納假設 $\rho(v_i, v_j) > k$ ，故 $\rho(v_i, v_j) = (A_{k+1})_{ij} = k+1$ ，由數學歸納法得證。 ■

若 Graph 為不連通圖，上述程序也必須在 A_{n-1} 時停下來，因為 $\text{diam}(G) \leq |V(G)| - 1$ 。我們將一般化作法寫成 Python 程式，詳見(附錄二)。

同心圓法、鄰接矩陣的元素加法都可以找到任兩點的距離，但若要計算最短路徑數就不容易從過程看出來，因此我們引入鄰接矩陣的乘法來處理這個問題。

(3) 鄰接矩陣的乘法

因為鄰接矩陣 A 的定義如下：若 v_i 與 v_j 相鄰，則 $A_{ij} = 1$ ，否則 $A_{ij} = 0$ 。

當 $A_{ij} = 0$ 表示從 v_i 到 v_j 不能只走一步就到， $A_{ij} = 1$ 表示可以只走一步就到，且方法數為1。

考慮 A 矩陣(鄰接矩陣)的次方 A^p ，

在 A^2 時， $(A^2)_{ij} = \sum_{k=1}^{n} A_{ik} A_{kj}$ ，若 $(A^2)_{ij} = \sum_{k=1}^{n} A_{ik} A_{kj} = r$ ，由於 $A_{ij} = 1 \vee 0$ ，這表示 $A_{ik} A_{kj} = 1$ 有 r 組，即 $\{v_i, v_k\}, \{v_k, v_j\} \in E(G)$ 有 r 組，故 $(A^2)_{ij}$ 表示 v_i 到 v_j 走 2 步到達的方法數有 r 種。

定理 3：v_i 到 v_j 走 p 步到達的方法數有 $(A^p)_{ij}$ 種。

證明：$(A^p)_{ij} = \sum_{n_1=1}^{n} (A^{p-1})_{in_1} A_{n_1 j} = \cdots = \sum_{n_1=1}^{n} \cdots \sum_{n_{p-1}=1}^{n} A_{in_{p-1}} A_{n_{p-1}n_{p-2}} \cdots A_{n_1 j}$ ，若 $(A^p)_{ij} = r$ ，由於 $A_{ij} = 1 \vee 0$ ，這表示 $A_{in_{p-1}} A_{n_{p-1}n_{p-2}} \cdots A_{n_1 j} = 1$ 有 r 組，即 $\{v_i, v_{n_{p-1}}\}, \{v_{n_{p-1}}, v_{n_{p-2}}\}, \cdots \{v_{n_1}, v_j\} \in E(G)$ 有 r 組，故 $(A^p)_{ij}$ 表示 v_i 到 v_j 走 p 步到達的方法數有 r 種。 ■

根據**定理 3**，若 m 為最小的自然數使得 $(A^m)_{ij} \neq 0$ ，這就表示，在連接 v_i, v_j 的路徑中，不存在長度小於 m 的路徑，且存在長度等於 m 的路徑(因為方法數不為 0)，即 $\rho(v_i, v_j) = m$ 。

一般化作法如下：

取每個 $(A^p)_{ij}$ 在最小次方時第一個不為 0 的值 $(=N_{ij})$，把這些取出的值組成一個新矩陣，此矩陣為「方法數矩陣 N」，此時的次方數 p 即為對應的最短步數，將次方數 p 取出放在第 i 列第 j 行所形成的矩陣為「距離矩陣 S」，也就是說考慮 A^p 在 (i,j) 元在 $p=1,2,3\cdots$ 的變化，則

$$S_{ij}=\min\{p\,|\,(A^p)_{ij}\neq 0\}\ ,\ N_{ij}=(A^{S_{ij}})_{ij}$$

因為回到自己的點不能算步數，所以 N 、 S 的主對角線皆為 0 。

事實上，當我們滿足 $A+A^2+\cdots+A^{|V(G)|-1}$ 的所有元皆不為 0 ，則為連通圖。我們將一般化作法寫成 Python 程式，詳見(附錄三)。

我們以圖 5 為例：

$$A^1=\begin{bmatrix}0&1&0&0&0&0&1&0&0\\1&0&0&0&1&0&0&1&0\\0&0&0&0&0&0&0&0&1\\0&0&0&0&1&0&1&0&0\\0&1&0&1&0&1&0&0&0\\0&0&0&0&1&0&0&0&1\\1&0&0&1&0&0&0&0&0\\0&1&\textcircled{0}&0&0&0&0&0&1\\0&0&1&0&0&1&0&1&0\end{bmatrix}\ (A^1)_{83}$$

$$A^2=\begin{bmatrix}2&0&0&1&1&0&0&1&0\\0&3&0&1&0&1&1&0&1\\0&0&1&0&0&1&0&1&0\\1&1&0&2&0&1&0&0&0\\1&0&0&0&3&0&1&1&1\\0&1&1&1&0&2&0&1&0\\0&1&0&0&1&0&2&0&0\\1&0&\textcircled{1}&0&1&1&0&2&0\\0&1&0&0&1&0&0&0&3\end{bmatrix}\ (A^2)_{83}$$

$$A^3=\begin{bmatrix}0&4&0&1&1&1&3&0&1\\4&0&1&1&5&1&1&4&1\\0&1&0&0&1&0&0&0&3\\1&1&0&0&4&0&3&1&1\\1&5&1&4&0&4&1&1&1\\1&1&0&0&4&0&1&1&4\\3&1&0&3&1&1&0&1&0\\0&4&\textcircled{0}&1&1&1&1&0&4\\1&1&3&1&1&4&0&4&0\end{bmatrix}\ (A^3)_{83}$$

$$A^4=\begin{bmatrix}7&1&1&4&6&2&1&5&1\\1&13&1&6&2&6&5&1&6\\1&1&3&1&1&4&0&4&0\\4&6&1&7&1&5&1&2&1\\6&2&1&1&13&1&5&6&6\\2&6&4&5&1&8&1&5&1\\1&5&0&1&5&1&6&1&2\\5&1&\textcircled{4}&2&6&5&1&8&1\\1&6&0&1&6&1&2&1&11\end{bmatrix}\ (A^4)_{83}$$

$$A^5=\begin{bmatrix}2&18&1&7&7&7&11&2&8\\18&4&6&7&25&8&7&19&8\\1&6&0&1&6&1&2&1&11\\7&7&1&2&18&2&11&7&8\\7&25&6&18&4&19&7&8&8\\7&8&1&2&19&2&7&7&17\\11&7&2&11&7&7&2&7&2\\2&19&\textcircled{1}&7&8&7&7&2&17\\8&8&11&8&8&17&2&17&2\end{bmatrix}\ (A^5)_{83}$$

$$N=\begin{bmatrix}0&1&1&1&1&1&1&1&1\\1&0&1&1&1&1&1&1&1\\1&1&0&1&1&1&2&1&1\\1&1&1&0&1&1&1&1&1\\1&1&1&1&0&1&1&1&1\\1&1&1&1&1&0&1&1&1\\1&1&2&1&1&1&0&1&2\\1&1&\textcircled{1}&1&1&1&1&0&1\\1&1&1&1&1&1&2&1&0\end{bmatrix}\quad N_{83}=(A^{S_{83}})_{83}=1$$

$$S=\begin{bmatrix}0&1&4&2&2&3&1&2&3\\1&0&3&2&1&2&2&1&2\\4&3&0&4&3&2&5&2&1\\2&2&4&0&1&2&1&3&3\\2&1&3&1&0&1&2&2&2\\3&2&2&2&1&0&3&2&1\\1&2&5&1&2&3&0&3&4\\2&1&\textcircled{2}&3&2&2&3&0&1\\3&2&1&3&2&1&4&1&0\end{bmatrix}\quad S_{83}=\min\{p\,|\,(A^p)_{83}\neq 0\}=2$$

(4) 各演算法的比較

通常要比較演算法的優劣，可以考慮將其寫為程式時，電腦所需要的執行時間。在電腦科學中，演算法的時間複雜度是一個函式，它描述了該演算法的執行時間。時間複雜度常用大O符號表述，即不包括這個函式的低階項和首項係數。

不失一般性，假設每個單元執行時間都是一樣的，那麼我們僅需考慮該演算法所需的操作單元數量。例如，對於一個演算法，如果輸入大小為n，它至多需要$5n^3+3n$個操作單元數量，那麼稱時間複雜度是$O(n^3)$。

在$k\times k$的遊戲地圖下：

同心圓法的操作單元數量估計：

每一個點要比對同行或同列共$2k$個點，每個點要比對2個元，並檢驗是否出現過，最多檢驗k^2個，共要執行k^2個頂點。又起始點的選擇有k^2個，故要找出任兩點的距離共要$(2k)\times 2\times k^2\times k^2\times k^2$個操作單元，所以時間複雜度是$O(k^7)$。

鄰接矩陣的元素加法：

距離矩陣需要考慮$\dfrac{k^2\times k^2-k^2}{2}$個待定的元，每個元需要檢查$k^2$次，此動作最多需執行$k^2-1$層，故需要$\dfrac{k^2\times k^2-k^2}{2}\times k^2\times(k^2-1)$個操作單元，所以時間複雜度是$O(k^8)$。

鄰接矩陣的乘法：

矩陣的乘法需要$2k^2\times k^2\times k^2$次操作單元，最多需要做k^2層，並比較這k^2層的每一個元，共$k^2\times k^2$個元，故需要$(2k^2\times k^2\times k^2)\times k^2\times k^2\times(k^2\times k^2)$個操作單元，所以時間複雜度是$O(k^{14})$。

演算法	同心圓法	鄰接矩陣的元素加法	鄰接矩陣的乘法
功能	任兩點最短距離 任兩點最短路徑	任兩點最短距離	任兩點最短距離 任兩點的最短路徑數
時間複雜度	$O(k^7)$	$O(k^8)$	$O(k^{14})$

對於給定的「遊戲地圖」，我們先將其轉換成圖論的 Graph，利用三種方法，計算出兩點的距離與最短路徑數。有了這些數值後，這讓我們開始思考，遊戲地圖是否有難易之別？

要界定遊戲難易度有許多面向可以考慮，我們選擇「任兩點的距離的平均值」來討論，我們將處理：什麼樣的「遊戲地圖」會有兩點平均距離的極值。

二、探討地圖任兩點平均距離的極值

底下的討論皆假定「遊戲地圖」有 k 個顏色與 k 個數字，故在 Graph 有 $n=k^2$ 個頂點。

(1) 平均距離的最小值

定理 4：若「遊戲地圖」的 (i,j) 位置放點 (i,j)，則該地圖相異兩點平均距離有最小值 $\frac{2k}{k+1}$。

證明：對於點 (i,j) 來說，能與它距離為1的點必形如 (i,p) 或 (p,j)，其餘的點與 (i,j) 的距離至少 2，故當地圖的 (i,j) 位置放點 (i,j)，則能與 (i,j) 距離為1的點，確實都是1，不能與 (i,j) 距離為1的點，則距離都是 2，故此地圖有平均距離最小值。因為在同行或同列皆可一步到達，其餘的格子走兩步即可抵達，相異兩點間的距離總和為

$$\left(2(k-1)\times 1+(k-1)^2\times 2\right)k^2=2k^3(k-1)$$

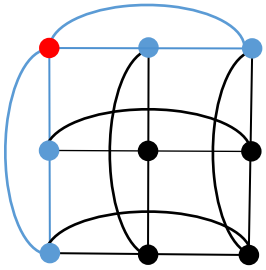
故平均距離為

$$\frac{2k^3(k-1)}{P_2^{k^2}}=\frac{2k^3(k-1)}{k^2(k+1)(k-1)}=\frac{2k}{k+1}$$

■

以 $k=3$ 為例，

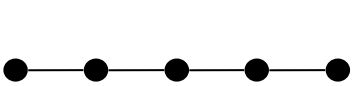
遊戲地圖：$\begin{bmatrix}(1,1) & (1,2) & (1,3)\\ (2,1) & (2,2) & (2,3)\\ (3,1) & (3,2) & (3,3)\end{bmatrix}$



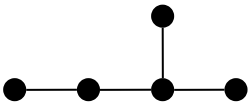
(2) 平均距離的最大值

與(1)平均距離的最小值的討論手法有些許不同，我們不容易直接猜測平均距離有最大值時的地圖長相，因此我們退一步，先考慮在 n 個點的 Graph 下，平均距離的最大值。

觀察下圖：



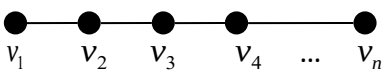
平均距離 2



平均距離 1.8

我們發現，在 Graph 為 P_n 時，兩點的平均距離似乎有比較大的趨勢，當有分支時，則會變小，故猜測在 n 個點的 Graph 下，平均距離的最大值發生在 P_n，且平均值為：

$$\frac{\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\rho(v_i,v_j)}{C_2^n}=\frac{\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}(k-i)}{C_2^n}=\frac{\sum_{i=1}^{n-1}\sum_{j=1}^{n-i}j}{C_2^n}=\frac{n+1}{3}$$



J.K. Doyle(1977) 針對此問題做更完整的討論為了方便討論，我們做底下定義：

在連通圖 $G(V,E)$ 中，$|V(G)|=n$ ，$\delta(G)=\sum_{u,v\in G}\rho(u,v)$ ，平均距離 $\mu(\mathrm{G})=\frac{\delta(G)}{C_2^n}$ 。

若 $v_i,v_j,v_k\in V$ 且滿足 $\rho(v_i,v_k)=\rho(v_i,v_j)+\rho(v_j,v_k)$ ，則稱 v_i,v_j,v_k 共線。

令 $\tau(G)=\left|\{\{v_i,v_j,v_k\}\mid v_i,v_j,v_k\text{不共線}\}\right|$

引理 (Doyle[2] , Proposition1.1)：

在連通圖 $G(V,E)$ 中，若 $|V(G)|=n$ ，且連接任兩點的最短路徑唯一，則

$$\mu(G)=\frac{n+1}{3}-\frac{\tau(G)}{C_2^n}\quad 。$$

證明：令 $C=\{\{v_i,v_j,v_k\}\mid v_i,v_j,v_k\in V(G)\text{, 且共線}\}$ ， $D=\{\{v_i,v_j\}\mid v_i,v_j\in V(G)\}$ ，

定義 $\varphi:C\to D$ 如下：

設 $\{v_1,v_2,v_3\}\in C$ ，其中 $\rho(v_1,v_3)=\rho(v_1,v_2)+\rho(v_2,v_3)$ ，則 $\varphi(\{v_1,v_2,v_3\})=\{v_1,v_3\}$ 。

因為最短路徑唯一，所以 $\varphi^{-1}(\{u,v\})=\{\{u,v,w\}\mid w\text{是連接}u,v\text{最短路徑中的頂點}\}$ ，

所以 $\left|\varphi^{-1}(\{u,v\})\right|=\rho(u,v)-1$ ， $|C|=\sum_{u,v\in V}(\rho(u,v)-1)=\delta(G)-C_2^n$ ，又 $|C|+\tau(G)=C_3^n$ ，

故 $\delta(G)=C_3^n+C_2^n-\tau(G)$ ，因此

$$\mu(G)=\frac{\delta(\mathrm{G})}{C_2^n}=\frac{n+1}{3}-\frac{\tau(G)}{C_2^n}$$

■

定理 5 (Doyle[2] , Corollary1.3**)**：

若 $G(V,E)$ 為 P_n ，則 Graph 的相異兩點平均距離有最大值 $\frac{(n+1)}{3}$ 。

證明：因為 $G(V,E)$ 為連通圖，令 G' 為其生成樹，

則 G' 滿足引理的條件要求(連接任兩點的最短路徑唯一)，且

$$\mu(G)\le\mu(G')=\frac{n+1}{3}-\frac{\tau(G')}{C_2^n}\le\frac{n+1}{3}$$

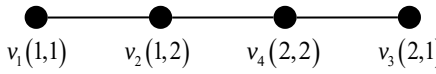
其中等號成立發生在 $G=G'$ 且 $\tau(G')=0$ ，而 $\tau(G')=0$ ，表示 $G'=P_n$ 。 ■

當發現平均距離的最大值為 $\frac{(n+1)}{3}$ 時，我們利用 Python「窮舉」可能的地圖設計。

雖然我們有找到 Graph 為 P_{16} 的遊戲地圖，但我們也發現到，按照我們的程式，當 $n=25$ 時，所費時間已超乎預期，因此我們必須另外考慮已知 Graph 下的遊戲地圖的設計方法。

由**定理 5**可知，在「Graph」上平均距離的最大值發生在P_n路徑，但我們要分清楚「Graph」與「遊戲地圖」的差異，我們舉例說明：

當$k=2$，$n=2\times 2$時，考慮 Graph 如下：



並不存在「遊戲地圖」使其「Graph」為上圖。

事實上，我們可以窮舉所有$k=2$連通的「遊戲地圖」

$$\begin{bmatrix}(1,1) & (1,2)\\ (2,1) & (2,2)\end{bmatrix}\begin{bmatrix}(1,2) & (2,2)\\ (1,1) & (2,1)\end{bmatrix}\begin{bmatrix}(2,2) & (2,1)\\ (1,2) & (1,1)\end{bmatrix}\begin{bmatrix}(2,1) & (1,1)\\ (2,2) & (1,2)\end{bmatrix}$$

$$\begin{bmatrix}(1,1) & (2,1)\\ (1,2) & (2,2)\end{bmatrix}\begin{bmatrix}(2,1) & (2,2)\\ (1,1) & (1,2)\end{bmatrix}\begin{bmatrix}(2,2) & (1,2)\\ (2,1) & (1,1)\end{bmatrix}\begin{bmatrix}(1,2) & (1,1)\\ (2,2) & (2,1)\end{bmatrix}$$

這些「遊戲地圖」的「Graph」都不是P_4，而這表示我們無法將任意的「Graph」找到「遊戲地圖」與之對應，但我們仍可提出一套從「Graph」回到「遊戲地圖」的方法，若過程中有產生矛盾則不存在對應的「遊戲地圖」，若無矛盾順利完成，則可以找到「遊戲地圖」與之對應。

三、提出滿足特殊條件的地圖之設計方法，並寫成 Python 程式，最後將其應用在地圖的極端情形(平均距離的極值)

「Graph」與「遊戲地圖」地圖最大的差異在於，「遊戲地圖」必須是在$k\times k$的方陣上，也就是遊戲有位置上的限制。為了建立「Graph」與「遊戲地圖」的關係，我們引入「位置矩陣P」、「卡牌矩陣C」的概念(底下皆以$k^2\times k^2$的矩陣，描述k^2個點之間的關係)。

(1) 「位置矩陣 P」、「卡牌矩陣 C」、「暫時位置矩陣 P^*」

卡牌矩陣C：若v_i、v_j的第 1 個元相同或第 2 個元相同，則$C_{ij}=1$，否則$C_{ij}=0$

位置矩陣P：若v_i、v_j的地圖位置在同一行或同一列，則$P_{ij}=1$，否則$P_{ij}=0$。

鄰接矩陣A：若v_i與v_j相鄰(「第 1 個元相同或第 2 個元相同」且「在同一行或同一列」)，則$A_{ij}=1$，否則$A_{ij}=0$。

我們從「遊戲地圖」轉換到「Graph」的頂點關係，我們要求：

「顏色相同或數字相同」且「地圖位置在同一行或同一列」

矩陣上，當$i\neq j$時，我們可以把上述關係寫成：

$$P_{ij}\times C_{ij}=A_{ij}$$

我們的目標是從「Graph」返回「遊戲地圖」，因此「Graph」的鄰接矩陣A已知，而卡牌矩陣C是固定的，所以我們可以反推P_{ij}，有了P_{ij}就相當於有頂點的位置關係。

但C_{ij}可能是0，故我們無法知道確切的「位置矩陣P」，因此，定義暫時位置矩陣P^*如下：

當$i=j$，$P^*_{ij}=1$。當$i\neq j$，若$C_{ij}\neq 0$，則$P^*_{ij}=\dfrac{A_{ij}}{C_{ij}}$，否則$P^*_{ij}=_$(待定數值)

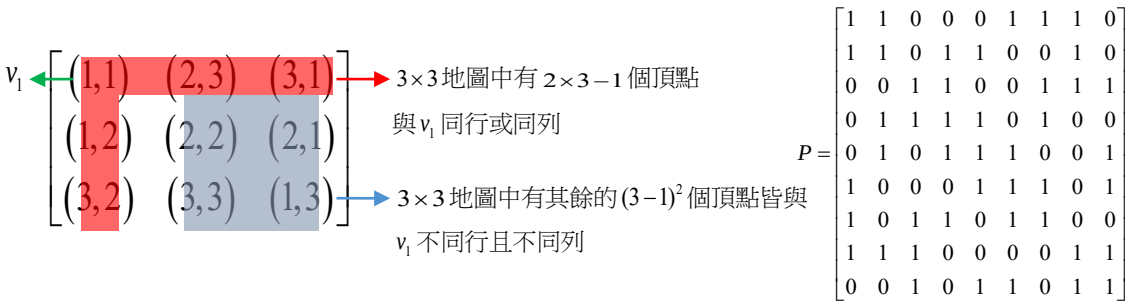
為了要將「暫時位置矩陣 P^* 」的待定數值找出來，必須充分了解「位置矩陣 P 」的性質。

(2) 在 k x k 的「地圖」中，「位置矩陣 P 」的性質

「位置矩陣 P 」必滿足：

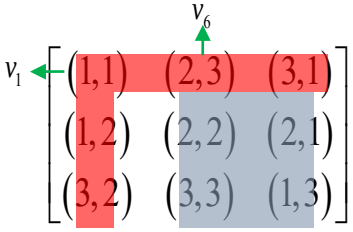
性質 1：因為 v_i 、 v_j 的「地圖位置在同一行或同一列」此條件有對稱性，所以 P 為對稱矩陣。

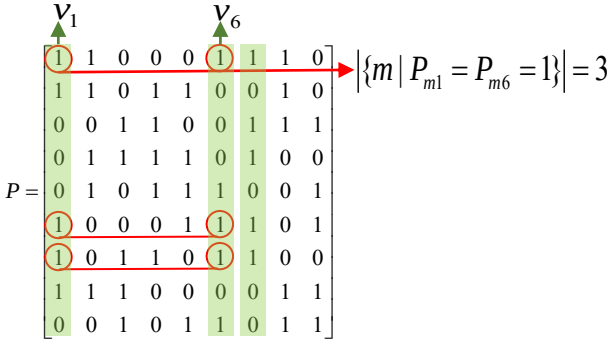
性質 2：在「地圖」中對於頂點 v_i ，必有 $2k-1$ 個頂點(包含自己)與 v_i 同行或同列，
其餘的 $(k-1)^2$ 個頂點皆與不同行且不同列，故 P 每一行的元中，
必有 $2k-1$ 個1， $(k-1)^2$ 個0 。



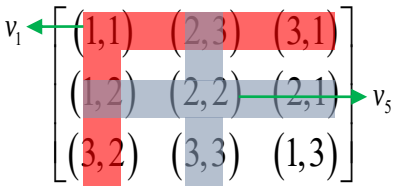
性質 3：在「地圖」中，若兩個頂點 v_i 、 v_j 落在同一行(或列)，則 v_i 、 v_j 同時與某 $k-2$ 個頂點落在同一行(或列)，故 P 的第 i 行與第 j 行將會恰好在相同的 k 個列中皆為1
即 $\left|\{m \mid P_{mi}=P_{mj}=1\}\right|=k$ 。

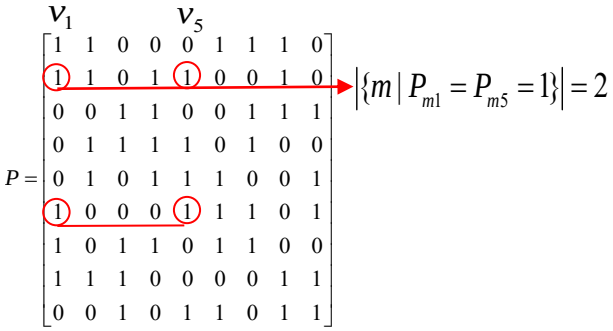
並且對任一個 v_i 而言，地圖上共有 $k-1$ 個頂點與 v_i 落在同一行(列)，
故 P 的第 i 行要與其他 $k-1$ 個行恰好在相同的 k 個列中皆為1
(並與其他另外 $k-1$ 個行恰好在相同的 k 個列中皆為1)。





性質 4：在「地圖」中，若兩個頂點v_i、v_j不落在同一行(列)，則v_i、v_j同時與某2個頂點落在同一行或同一列，故P的第i行與第j行將會恰好在相同的2個列中皆為1，即$\left|\{m \mid P_{mi}=P_{mj}=1\}\right|=2$，並且對任一個$v_i$而言，地圖上共有$(k-1)^2$個頂點與$v_i$不落在同一行且不落在同一列，故$P$的第$i$行要與其他$(k-1)^2$個行恰好在相同的2個列中皆為1。





也就是性質 1~4 為某「地圖」的「位置矩陣P」之必要條件，而P^2可以體現性質 1~4，理由如下：

$$(P^2)_{ij}=\sum_{k=1}^{n}P_{ik}P_{kj}\ ，若(P^2)_{ij}=\sum_{k=1}^{n}P_{ik}P_{kj}\overset{P為對稱矩陣}{=}\sum_{k=1}^{n}P_{ki}P_{kj}=r\ ，$$

由於$P_{ij}=1\vee 0$，這代表第i行與第j行有r個位置有共同的1。

故性質 1，P^2為對稱矩陣；性質 2，$(P^2)_{ii}=2k-1$；性質 3 及性質 4，$(P^2)_{ij}=2\vee k\ \ \forall i\neq j$。

例如：

$$P=\begin{bmatrix}1&1&0&0&0&1&1&1&0\\1&1&0&1&1&0&0&1&0\\0&0&1&1&0&0&1&1&1\\0&1&1&1&1&0&1&0&0\\0&1&0&1&1&1&0&0&1\\1&0&0&0&1&1&1&0&1\\1&0&1&1&0&1&1&0&0\\1&1&1&0&0&0&0&1&1\\0&0&1&0&1&1&0&1&1\end{bmatrix}\qquad P^2=\begin{bmatrix}5&3&2&2&2&3&3&3&2\\3&5&2&3&3&2&2&3&2\\2&2&5&3&2&2&3&3&3\\2&3&3&5&3&2&3&2&2\\2&3&2&3&5&3&2&2&3\\3&2&2&2&3&5&3&2&3\\3&2&3&3&2&3&5&2&2\\3&3&3&2&2&2&2&5&3\\2&2&3&2&3&3&2&3&5\end{bmatrix}$$

事實上，我們可以說明，性質 1~4 亦是「地圖」的「位置矩陣P」的充分條件，如底下的**定理 6**。

定理 6：若$Q_{k^2\times k^2}$滿足性質 1~4，則存在「遊戲地圖」使其該地圖的「位置矩陣」為$Q_{k^2\times k^2}$。

亦即，$Q_{k^2\times k^2}$是某「遊戲地圖」的「位置矩陣P」。

證明：目標是找到「遊戲地圖」，因為Q是$k^2\times k^2$的矩陣，我們可以把Q_{ij}想成頂點v_i與v_j的關係，即$Q_{ij}=1$代表v_i與v_j要在同一行或同一列，而$Q_{ij}=0$代表v_i與v_j不在同一行且不在同一列。

先將v_1置於棋盤(1,1)位置，

考慮第1行，根據性質 2可以找到$2(k-1)$個頂點與之同行或同列，再由性質 3，我們可從中區分成$k-1$個頂點與之同行，故將這$k-1$個頂點隨機置入棋盤的第一行，

不妨假設為，$v_{g(2)},v_{g(3)},\cdots,v_{g(k)}$。

另外$k-1$個頂點與之同列，亦將這$k-1$個頂點隨機置入棋盤的第一列，

不妨假設為，$v_{f(2)},v_{f(3)},\cdots,v_{f(k)}$。

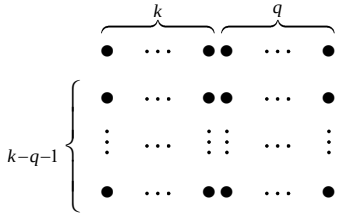
即：

v_1	$v_{f(2)}$	$v_{f(3)}$	$\cdots$	$v_{f(k)}$
$v_{g(2)}$				
$v_{g(3)}$				
$\vdots$				
$v_{g(k)}$				

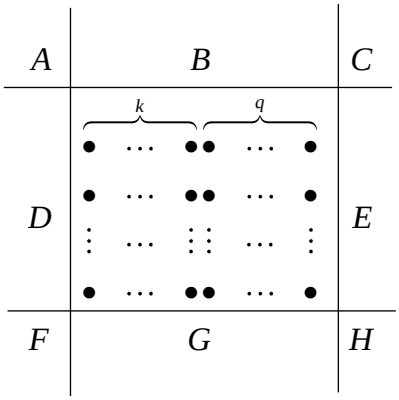
考慮棋盤(i,j)位置$i>1$、$j>1$，因為$v_{g(i)}$與$v_{f(j)}$不在同一行，也不在同一列，根據性質 4，可找到2個頂點與之同行或同列，當然其中一個是v_1，另一個則填在棋盤(i,j)位置。 ■

事實上，在$k^2\times k^2$的棋盤上若區域的第i行第j列有頂點，第i行(j列)必恰有k個頂點，也就是說，所有頂點的位置會形成$k\times k$的地圖。

這是因為，若棋盤第j列的頂點數為p且$p>k$，不妨假設$p=k+q\,(0<q<k-1)$，根據性質 2，第i行必須再補足$k-q-1$個點，同理第j列上有頂點的行也都要補足$k-q-1$個點。因此我們會用掉$(k+q)(k-q-1+1)=k^2-q^2$個頂點。



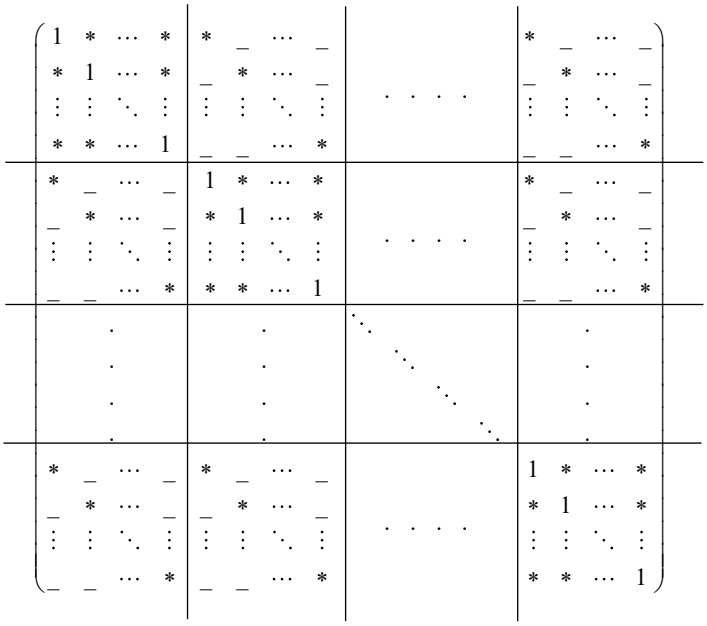
因為 $q>0$，所剩的點 $q^2>0$，如下圖所示，剩下的點必不能放在區域 B,D,G,E，否則會違背 性質 2，若放在區域 A,C,F,H，又與 性質 4 矛盾，故 $q=0$，得證 $p=k$，即所有頂點的位置會形成 $k\times k$ 的方陣。



定理 6 除了告訴我們滿足性質 1~4 的 $Q_{k^2\times k^2}$ 是某「遊戲地圖」的「位置矩陣 P」，其證明過程亦提供了一套由「位置矩陣 P」回到「遊戲地圖」的方法。

(3) 「暫時位置矩陣 P^*」反推「位置矩陣 P」

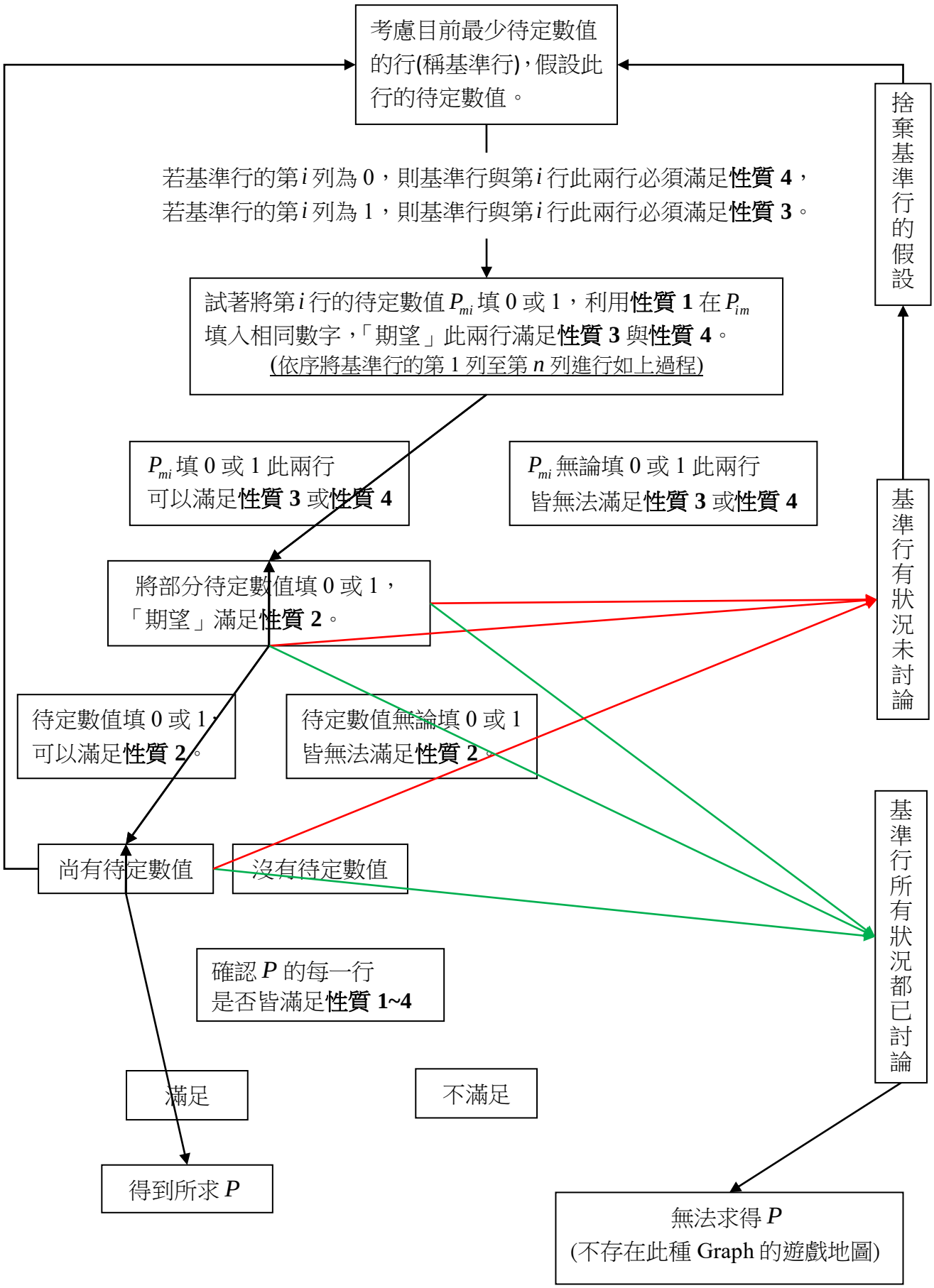
給定 $P^*_{k^2\times k^2}$，因為卡牌矩陣 0 的位置固定，所以 P^* 不確定的位置也固定，故 $P^*_{k^2\times k^2}$ 形如：



其中 $*$ 會依據給定的 Graph 確定 0 或 1，而 _ 為待定數值(0 或 1)。

我們將會給出一套從「暫時位置矩陣 P^*」推得「位置矩陣 P」的過程，並給兩個例子分別說明有些 P^* 可以推得 P，而有些則不行。

「暫時位置矩陣 P^*」推得「位置矩陣 P」的流程圖：



例子一：

$$P^*=\begin{bmatrix}1&1&0&0&\_&\_&0&\_&\_\\1&1&1&\_&0&\_&\_&0&\_\\0&1&1&\_&\_&0&\_&\_&1\\0&\_&\_&1&0&1&1&\_&\_\\\_&0&\_&0&1&0&\_&1&\_\\\_&\_&0&1&0&1&\_&\_&1\\0&\_&\_&1&\_&\_&1&1&0\\\_&0&\_&\_&1&\_&1&1&0\\\_&\_&1&\_&\_&1&0&0&1\end{bmatrix}$$

從基準行(第 1 行)開始將底線依序假設為 1,0,1,1：

$$\begin{bmatrix}1&1&0&0&1&0&0&1&1\\1&1&1&\_&0&\_&\_&0&1\\0&1&1&\_&0&0&\_&0&1\\0&\_&\_&1&0&1&1&\_&\_\\1&0&0&0&1&0&\_&1&0\\0&\_&0&1&0&1&\_&\_&1\\0&\_&\_&1&\_&\_&1&1&0\\1&0&0&\_&1&\_&1&1&0\\1&1&1&\_&0&1&0&0&1\end{bmatrix}$$

依據性質 2，產生矛盾。

1. 由性質 1 得 $(P^*)_{15}=1,(P^*)_{16}=0,(P^*)_{18}=1,(P^*)_{19}=1$
2. $(P^*)_{21}=1$，由性質 3 知第 1 行與第 2 行要在某 3 列同時為 1，知 $(P^*)_{92}=1$；再由性質 1 得 $(P^*)_{29}=1$。
3. $(P^*)_{31}=0$，由性質 4 知第 1 行與第 3 行要在某 2 列同時為 1，知 $(P^*)_{53}=0$，$(P^*)_{83}=0$；
   再由性質 1 得 $(P^*)_{35}=0$，$(P^*)_{38}=0$。
4. $(P^*)_{41}=0$，由性質 4 知第 1 行與第 4 行要在某 2 列同時為 1，無法由性質確定，保持待定_。
5. $(P^*)_{51}=1$，由性質 3 知第 1 行與第 5 行要在某 3 列同時為 1，無法由性質確定，保持待定_。
6. $(P^*)_{61}=0$，由性質 3 知第 1 行與第 6 行要在某 3 列同時為 1，無法由性質確定，保持待定_。
7. $(P^*)_{71}=0$，由性質 4 知第 1 行與第 7 行要在某 2 列同時為 1，無法由性質確定，保持待定_。
8. $(P^*)_{81}=1$，由性質 3 知第 1 行與第 8 行要在某 3 列同時為 1，無法由性質確定，保持待定_。
9. $(P^*)_{91}=1$，由性質 3 知第 1 行與第 9 行要在某 3 列同時為 1，無法由性質確定，保持待定_。
10. 檢查各行 1 的各數，發現第 5 行無法與性質 2 產生矛盾。

捨棄基準行(第 1 行)1,1,0,1 的假設，將基準行(第一行)底線依序假設為 0,1,1,1

仿造上述過程可得：

$$\begin{bmatrix}1&1&0&0&0&1&0&1&1\\1&1&1&\_&0&0&1&0&\_\\0&1&1&\_&\_&0&\_&0&1\\0&\_&\_&1&0&1&1&\_&\_\\0&0&\_&0&1&0&\_&1&1\\1&0&0&1&0&1&1&0&1\\0&1&\_&1&\_&1&1&1&0\\1&0&0&\_&1&0&1&1&0\\1&\_&1&\_&1&1&0&0&1\end{bmatrix}$$

依據性質 3，產生矛盾。

捨棄基準行(第 1 行)0,1,1,1 的假設，將基準行(第一行)底線依序假設為 1,1,0,1

仿造上述過程可得：

$$\begin{bmatrix}1&1&0&0&1&1&0&0&1\\1&1&1&\_&0&0&\_&0&\_\\0&1&1&\_&0&0&\_&\_&1\\0&\_&\_&1&0&1&1&\_&\_\\1&0&0&0&1&0&\_&1&1\\1&0&0&1&0&1&\_&1&1\\0&\_&\_&1&\_&\_&1&1&0\\0&0&\_&\_&1&1&1&1&0\\1&\_&1&\_&1&1&0&0&1\end{bmatrix}$$

依據性質 3，產生矛盾。

捨棄基準行(第 1 行)1,1,0,1 的假設，將基準行(第一行)底線依序假設為 1,1,1,0

仿造上述過程可得：

$$\begin{bmatrix}1&1&0&0&1&1&0&1&0\\1&1&1&0&0&1&1&0&0\\0&1&1&0&1&0&1&0&1\\0&0&0&1&0&1&1&1&1\\1&0&1&0&1&0&0&1&1\\1&1&0&1&0&1&0&0&1\\0&1&1&1&0&0&1&1&0\\1&0&0&1&1&0&1&1&0\\0&0&1&1&1&1&0&0&1\end{bmatrix}$$

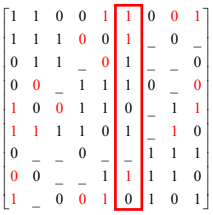
此矩陣滿足 P 的性質，故為解。

例子二：

$$P^* = \begin{bmatrix} 1 & 1 & 0 & 0 & \_ & \_ & 0 & \_ & \_ \\ 1 & 1 & 1 & \_ & 0 & \_ & \_ & 0 & \_ \\ 0 & 1 & 1 & \_ & \_ & 1 & \_ & \_ & 0 \\ 0 & \_ & \_ & 1 & 1 & 1 & 0 & \_ & \_ \\ \_ & 0 & \_ & 1 & 1 & 0 & \_ & 1 & \_ \\ \_ & \_ & 1 & 1 & 0 & 1 & \_ & \_ & 0 \\ 0 & \_ & \_ & 0 & \_ & \_ & 1 & 1 & 1 \\ \_ & 0 & \_ & \_ & 1 & \_ & 1 & 1 & 0 \\ \_ & \_ & 0 & \_ & \_ & 0 & 1 & 0 & 1 \end{bmatrix}$$

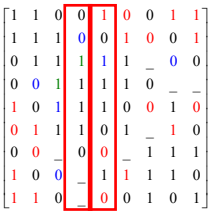
從第一行開始將底線依序假設為 1,1,1,0：

捨棄基準行(第一行)1,1,1,0 的假設，將基準行(第一行)底線依序假設為 1,1,0,1



依據性質 2，產生矛盾。

捨棄基準行(第一行) 1,1,0,1 的假設，將基準行(第一行)底線依序假設為 1,0,1,1



依據性質 3，產生矛盾。

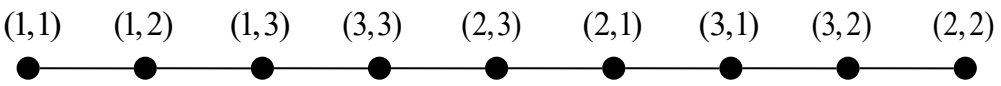
捨棄基準行(第一行) 1,0,1,1 的假設，將基準行(第一行)底線依序假設為 0,1,1,1

依據性質 3，產生矛盾。

根據上述討論，此 P^* 無法找到遊戲地圖與之對應。

為了找到平均距離有最大值的「遊戲地圖」，我們利用(1)(2)(3)提出地圖設計方法，首先，根據頁 15 **定理 5**，當 $k=3,n=9$ 時， P_9 有平均距離最大值 $\frac{10}{3}$ 。其次，我們利用(2)(3)尋找滿足 Graph 為 P_9 的「遊戲地圖」。

首先，考慮 P_9 的其中一種排序如下：



此 Graph 的「鄰接矩陣 A 」為

$$A=\begin{bmatrix}0&1&0&0&0&0&0&0&0\\1&0&1&0&0&0&0&0&0\\0&1&0&0&0&0&0&0&1\\0&0&0&0&0&1&1&0&0\\0&0&0&0&0&0&0&1&0\\0&0&0&1&0&0&0&0&1\\0&0&0&1&0&0&0&1&0\\0&0&0&0&1&0&1&0&0\\0&0&1&0&0&1&0&0&0\end{bmatrix}$$

為了推算頂點在「遊戲地圖」的位置，我們利用(1)計算「暫時位置矩陣 P^* 」

$$P^*=\begin{bmatrix}1&1&0&0&\_&\_&0&\_&\_\\1&1&1&\_&0&\_&\_&0&\_\\0&1&1&\_&\_&0&\_&\_&1\\0&\_&\_&1&0&1&1&\_&\_\\\_&0&\_&0&1&0&\_&1&\_\\\_&\_&0&1&0&1&\_&\_&1\\0&\_&\_&1&\_&\_&1&1&0\\\_&0&\_&\_&1&\_&1&1&0\\\_&\_&1&\_&\_&1&0&0&1\end{bmatrix}$$

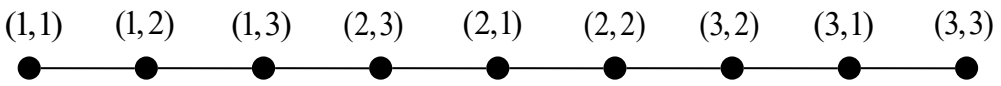
再根據(3)的流程，我們可以推算出「位置矩陣 P 」

$$P=\begin{bmatrix}1&1&0&0&1&1&0&1&0\\1&1&1&0&0&1&1&0&0\\0&1&1&0&1&0&1&0&1\\0&0&0&1&0&1&1&1&1\\1&0&1&0&1&0&0&1&1\\1&1&0&1&0&1&0&0&1\\0&1&1&1&0&0&1&1&0\\1&0&0&1&1&0&1&1&0\\0&0&1&1&1&1&0&0&1\end{bmatrix}$$

定理 6的證明過程提供了一套由「位置矩陣 P 」回到「遊戲地圖」的方法，故可將上述的 P 矩陣，轉回遊戲地圖如：

$$\begin{bmatrix}(1,1)&(1,2)&(2,3)\\(3,2)&(3,1)&(2,1)\\(2,2)&(1,3)&(3,3)\end{bmatrix}$$

注意到，我們一樣考慮 P_9 Graph，但重新配置頂點的連接順序如下：



(此 Graph 的暫時位置矩陣 P^* 即為頁 23 的例子 2)

根據頁 23 的討論此時的 P^* 無法回到 P ，這表示在這樣順序下的 Graph，並不存在「遊戲地圖」與之對應。

至此我們得到了將給定的 Graph 放進遊戲地圖的方法，我們將上述過程撰寫 Python 程式，詳見(附錄四)。

一般來說，我們可以考慮「遊戲地圖」上的各種參數(本研究著重在平均距離)，而當我們計算這些參數的極值，某些時候在 Graph 上會比在「遊戲地圖」上簡單，而且有許多 Graph 理論提供分析，因此我們可以考慮 Graph 上的參數，再想辦法找到「遊戲地圖」來對應 Graph。

同理，要設計 Graph 為 P_{k^2} 的 $k\times k$「遊戲地圖」，我們可以考慮各種連接順序的 P_{k^2} Graph，利用本研究的方法得到「遊戲地圖」。

柒、研究結論

一、給定「遊戲地圖」，將其轉換成 Graph，並將 Graph 記錄成「鄰接矩陣 A」

(1) 將 Graph 轉為同心圓，可以從同心圓上讀出距離與路徑。

(2) 利用「鄰接矩陣 A」的特殊加法，也可以計算任兩點的距離。

(3) 利用「鄰接矩陣 A」的乘法，可以計算任兩點的距離，並求得最短路徑數。

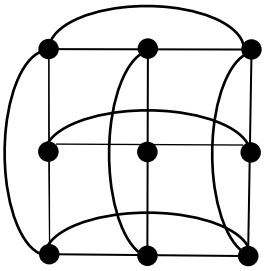
(4) 比較三種演算法的功能與時間複雜度：

演算法	同心圓法	鄰接矩陣的元素加法	鄰接矩陣的乘法
功能	任兩點最短距離 任兩點最短路徑	任兩點最短距離	任兩點最短距離 任兩點的最短路徑數
時間複雜度	$O(k^7)$	$O(k^8)$	$O(k^{14})$

二、地圖平均距離的極值

(1) k 個顏色，k 個數字的遊戲地圖，平均距離的最小值 $\frac{2k}{k+1}$，且在數字表徵下的遊戲地圖為地圖的 (i,j) 位置放點 (i,j)。

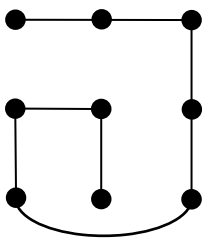
$$\begin{bmatrix} (1,1) & (1,2) & (1,3) \\ (2,1) & (2,2) & (2,3) \\ (3,1) & (3,2) & (3,3) \end{bmatrix}$$



(2) k 個顏色，k 個數字的遊戲地圖，平均距離的最大值不易直接猜測地圖長相，

我們先考慮，在 n 點的 Graph 下，其平均距離的最大值為 $\frac{n+1}{3}$，且 Graph 必為 P_n。

$$\begin{bmatrix} (1,1) & (1,2) & (2,2) \\ (3,2) & (3,1) & (2,3) \\ (3,3) & (2,1) & (1,3) \end{bmatrix}$$



三、探討給定 Graph 下，遊戲地圖設計的方法與可行性

(1) 將地圖設計的問題轉換成矩陣形式。

(2) 遊戲地圖的「位置矩陣」具有以下的性質：

1. P 是對稱矩陣。
2. P 的每一行都有 $2k-1$ 個 1，$(k-1)^2$ 個 0。
3. P 的第 i 行與第 j 行恰好在相同的 k 個列中皆為 1，即 $\left|\{m \mid P_{mi}=P_{mj}=1\}\right|=k$。

   並且對任一個 v_i 而言，地圖上共有 $k-1$ 個頂點與 v_i 落在同一行(列)，

   故 P 的第 i 行要與其他 $k-1$ 個行恰好在相同的 k 個列中皆為 1。

   (並與其他另外 $k-1$ 個行恰好在相同的 k 個列中皆為 1)
4. P 的第 i 行與第 j 行將會恰好在相同的 2 個列中皆為 1，即 $\left|\{m \mid P_{mi}=P_{mj}=1\}\right|=2$，

   並且對任一個 v_i 而言，地圖上共有 $(k-1)^2$ 個頂點與 v_i 不落在同一行且不落在同一列，故 P 的第 i 行要與其他 $(k-1)^2$ 個行恰好在相同的 2 個列中皆為 1。

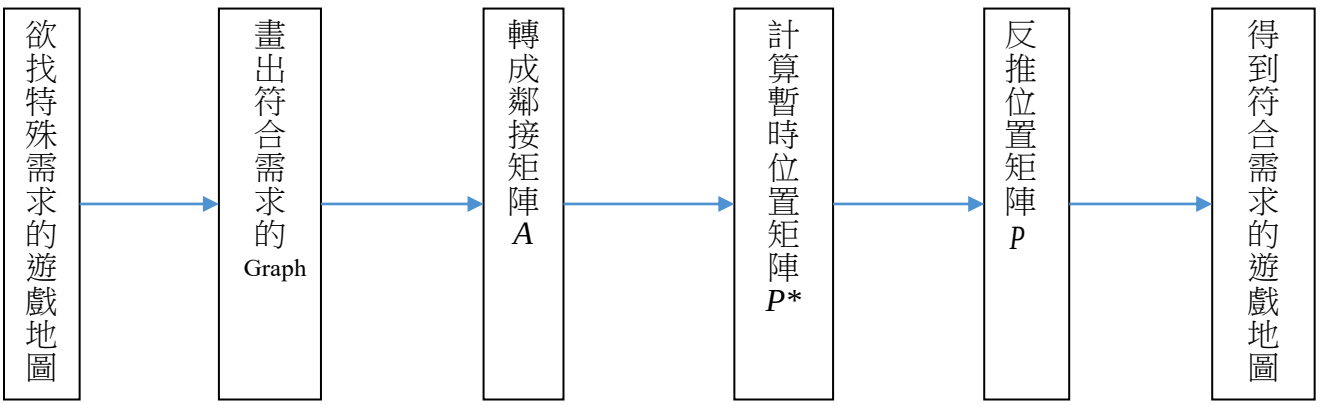
反之，滿足上述性質的矩陣，也是某遊戲地圖的位置矩陣。

(3) 給出一套從「暫時位置矩陣 P^*」推得「位置矩陣 P」的過程(頁 21)，並推得原始地圖。

總結來說，研究結果一提供了我們對於任一個遊戲地圖的分析手法，以計算「任兩點的最短距離」、「最短路徑」與「最短路徑數」。

當我們有不同的遊戲地圖需求時，像是平均距離最大或最小(如研究結果二)、在地圖上隨機移動能到達的平均機率最大或最小、最短路徑的平均數量最大或最小等等……。無論如何，我們都可以先進行類似研究結果二的方式分析，將遊戲地圖轉為 Graph，考慮這些 Graph 的特質，進而得到不同地圖參數出現極值時，所擁有的地圖特質。

研究結果三主要處理將給定的 Graph 放進遊戲地圖的方法：



因此 P_n Graph 的遊戲地圖存在性只是此研究中的一種特例，往後我們可以針對不同的地圖需求，結合研究結果一二三，達到地圖設計或驗證存在性的目的。

捌、未來展望

一、設 $n=k\times k$ ，證明必存在 Graph 為 P_n 的 $k\times k$ 遊戲地圖， $\forall k\geq 3$ 。(已知 $k=3,4$ 均存在)

想法：找出 Graph 為 P_n 的位置矩陣 P 之特性。

二、定義其他的遊戲地圖參數，討論各種參數極值下的地圖設計。

想法：在遊戲地圖上隨機移動能到達的平均機率、最短路徑的平均數量極值的地圖設計。

三、給定 Graph 下的3維遊戲地圖設計，進而推廣至 n 維遊戲地圖設計，並寫成程式。

想法：考慮 $V(G)=\{(x,y,x)\,|\,1\leq x,y,z\leq k\}$ ，定義 A,C,P,P^* ，並分析其關係。

玖、參考文獻

1.邵慰慈、潘建強(民 94)。**基礎離散數學**。台北市：九章出版社。

2. J.K. Doyle(1977). Mean distance in a graph, Discrete Mathematics. Volume 17, Issue 2, 1977, Pages 147-154.

拾、附錄

附錄一、同心圓法 Python 程式碼

```
def findPointLoc(point, xmap):
    for i in range(len(xmap)):
        for j in range(len(xmap[0])):
            if xmap[i][j] == point:
                return (i, j)
    return (-1,-1)
def repeatCheck(point, passed):
    if point in passed:
        return True
    return False
def beNext(pointx, pointy):
    if pointx[0:1] == pointy[0:1] or pointx[1:2] == pointy[1:2]:
        return True
    return False
def routeRtn(point, xmap, passed):
    xpoint = findPointLoc(point, xmap)
    ans = []
    for i in range(len(xmap)):
        if not repeatCheck(xmap[i][xpoint[1]], passed):
            if beNext(point, xmap[i][xpoint[1]]):
                ans += [(point, xmap[i][xpoint[1]])]
                passed += [xmap[i][xpoint[1]]]
    for j in range(len(xmap)):
        if not repeatCheck(xmap[xpoint[0]][j], passed):
            if beNext(point, xmap[xpoint[0]][j]):
                ans += [(point, xmap[xpoint[0]][j])]
                passed += [xmap[xpoint[0]][j]]
    return ans
def moveRtn(xmap, route, passed):
    endw = True
    while endw:
        nxtRoute = []
        for x in route[-1]:
            nxtRoute += routeRtn(x[1], xmap, passed)
        if len(nxtRoute) > 0:
            route += [nxtRoute]
        else:
            endw = False
    else:
        return route
def main(point):
    map=[["G1","R3","W1"],["G2","R2","R1"],["W2","W3","G3"]]
    passed = [point]
    route = [[(passed[0], passed[0])]]
    route = moveRtn(map, route, passed)
    print("路徑圖:")
    for x in map:
        print(x)
    print(point + "  可到達的點數: " , len(passed)-1)
    print(point + "至各點的步數及路徑:")
    n=0
    for x in route:
        print("需", n, "步:", x)
        n += 1
print("----------     End of Program!    ----------")
```

執行結果

```
>>> main("G3")
路徑圖:
['G1', 'R3', 'W1']
['G2', 'R2', 'R1']
['W2', 'W3', 'G3']
G3  可到達的點數:   8
G3 至各點的步數及路徑:
需  0  步: [('G3', 'G3')]
需  1  步: [('G3', 'W3')]
需  2  步: [('W3', 'R3'), ('W3', 'W2')]
需  3  步: [('R3', 'R2'), ('W2', 'G2')]
需  4  步: [('R2', 'R1'), ('G2', 'G1')]
需  5  步: [('R1', 'W1')]
----------     End of Program!    ----------
```

附錄二、鄰接矩陣的元素加法 Python 程式碼

```
def distancek(A,k):
    C = [[0 for i in range(len(A))]for j in range(len(A))]
    for i in range(len(A)):
        for j in range(len(A)):
            if i!=j and 0 < A[i][j] < k:
                C[i][j]=A[i][j]
            elif i!=j and A[i][j]==0:
                for p in range(len(A)):
                    if A[i][p]+A[p][j]==k:
                        C[i][j]=k
    return(C)
def distancematrix(map):
    C = adjmatrix(map)
    for i in range(len(adjmatrix(map))):
        C = distancek(C,i+2)
    return C
```

執行結果

```
路徑圖
[(1, 1), (2, 3), (3, 1)]
[(1, 2), (2, 2), (2, 1)]
[(3, 2), (3, 3), (1, 3)]
距離矩陣
[0, 1, 4, 2, 2, 3, 1, 2, 3]
[1, 0, 3, 2, 1, 2, 2, 1, 2]
[4, 3, 0, 4, 3, 2, 5, 2, 1]
[2, 2, 4, 0, 1, 2, 1, 3, 3]
[2, 1, 3, 1, 0, 1, 2, 2, 2]
[3, 2, 2, 2, 1, 0, 3, 2, 1]
[1, 2, 5, 1, 2, 3, 0, 3, 4]
```

附錄三、鄰接矩陣乘法 Python 程式碼

```
def solRtn(A, n):
    M = [A]
    B = [[A[i][j] for j in range(len(A))] for i in range(len(A))]
    for i in range(n):
        B = A2(A, B)
        M += [B]
    AA = [[A[i][j] for j in range(len(A))] for i in range(len(A))]
    BB = [[A[i][j] for j in range(len(A))] for i in range(len(A))]
    for k in range(len(M)):
        for i in range(len(M[k])):
            for j in range(len(M[k])):
                if i != j and AA[i][j] == 0 and M[k][i][j] > 0:
                    AA[i][j] = M[k][i][j]
                    BB[i][j] = k+1
def A2(A, B):
    C = [[0 for x in range(len(A))] for y in range(len(A))]
    for i in range(len(A)):
        for j in range(len(A[i])):
            for k in range(len(A)):
                C[i][j] += A[i][k] * B[k][j]
    return C
def adjmatrix(map):
    C = [[0 for x in range(len(map[0])**2)] for y in
        range(len(map[0])**2)]
    for i in range(len(map[0])):
        for j in range(len(map[0])):
            x =(map[i][j][0]-1)*(len(map[0])) + map[i][j][1] - 1
            for m in range(len(map[0])):
                if map[m][j][0] == map[i][j][0] or
                   map[m][j][1] == map[i][j][1]:
                    y = (map[m][j][0]-1)*(len(map[0])) +
                        map[m][j][1] - 1
                    C[x][y] = 1
            for n in range(len(map[0])):
                if map[i][n][0] == map[i][j][0] or
                   map[i][n][1] == map[i][j][1]:
                   y = (map[i][n][0]-1)*(len(map[0]))+map[i][n][1] - 1
                    C[x][y] = 1
    for k in range(len(map[0])**2):
        C[k][k] = 0
    return(C)
```

執行結果

```
>>>D = [[(1,1),(2,3),(3,1)],[(1,2),(2,2),(2,1)],[(3,2),(3,3),(1,3)]]
    solRtn(adjmatrix(D), len(D[0])**2)
        方法數矩陣:                    距離矩陣:
        [0, 1, 1, 1, 1, 1, 1, 1, 1]    [0, 1, 4, 2, 2, 3, 1, 2, 3]
        [1, 0, 1, 1, 1, 1, 1, 1, 1]    [1, 0, 3, 2, 1, 2, 2, 1, 2]
        [1, 1, 0, 1, 1, 1, 2, 1, 1]    [4, 3, 0, 4, 3, 2, 5, 2, 1]
        [1, 1, 1, 0, 1, 1, 1, 1, 1]    [2, 2, 4, 0, 1, 2, 1, 3, 3]
        [1, 1, 1, 1, 0, 1, 1, 1, 1]    [2, 1, 3, 1, 0, 1, 2, 2, 2]
        [1, 1, 1, 1, 1, 0, 1, 1, 1]    [3, 2, 2, 2, 1, 0, 3, 2, 1]
        [1, 1, 2, 1, 1, 1, 0, 1, 2]    [1, 2, 5, 1, 2, 3, 0, 3, 4]
        [1, 1, 1, 1, 1, 1, 1, 0, 1]    [2, 1, 2, 3, 2, 2, 3, 0, 1]
        [1, 1, 1, 1, 1, 1, 2, 1, 0]    [3, 2, 1, 3, 2, 1, 4, 1, 0]
```

附錄四、地圖設計 Python 程式碼

```
import math
import numpy
import itertools
    def combinations(dim, mincnt, cnt1):
        A = [i for i in range(mincnt)]
        C = []
        for x in itertools.combinations(A, dim - cnt1):
            D = [0 for i in range(mincnt)]
            for d in x:
                D[d] = 1
            C += [D]
        return C

    def countn(Ak, n):
        initial = []
        for i in range(len(Ak)):
            if Ak[i] == n:
                initial = initial + [i]
        return initial

    def count(Ak,n):
        ans = 0
        for x in Ak:
            if x == n:
                ans += 1
        return ans

    def countcompare(Ak,Al):
        C = [0 for i in range(len(Ak))]
        for i in range(len(Ak)):
            C[i] = Ak[i]*Al[i]
        return C

    def compare(Ak,Al,m):
        C = [Al[i] for i in range(len(Ak))]
        if count(countcompare(Ak,Al),1) == m:
            for x in countn(Al,3):
                if Ak[x] == 1:
                    C[x] = 0
        elif count(countcompare(Ak,Al),1) < m:
            for x in countn(Al,3):
                if count(countcompare(Ak,Al),3) <= (m -
count(countcompare(Ak,Al),1) )and Ak[x] == 1:
                    C[x] = 1
                else:
                    continue
        return C

    def pstarcheck(C):
        dim = 2*int(len(C)**0.5)-1
        for k in range(len(C)):
            if count(C[k],1) > dim:
                return False
            if (count(C[k],3) + count(C[k],1)) < dim:
                return False
        return True

    def rowfill(A, nrow, list1):
        n = 0
        C = [[A[i][j] for j in range(len(A))] for i in range(len(A))]
        for i in range(len(A)):
            if C[nrow][i] == 3 :
                C[nrow][i] = list1[n]
                C[i][nrow] = list1[n]
                n = n +1
        return C

    def max3(A):
        ndx = 0
        maxcnt = -1
        for i in range(len(A)):
            cnt3 = 0
            for j in range(len(A)):
                if A[i][j] == 3 :
                    cnt3 += 1
            if cnt3 > maxcnt:
                ndx = i
                maxcnt = cnt3
        cnt1 = 0
        for i in range(len(A)):
            if A[ndx][i] == 1:
                cnt1 += 1
        return(ndx, maxcnt, cnt1)

    def modify3(A):
        C = [[A[i][j] for j in range(len(A))] for i in range(len(A))]
        dim = 2*int(len(A)**0.5)-1
        for i in range(len(A)):
            cnt1 = 0
            for j in range(len(A)):
                if A[i][j] == 1 :
                    cnt1 += 1
            if cnt1 == dim :
                for j in range(len(A)):
                    if A[i][j] == 3:
                        A[i][j] = 0
            if cnt1 + count(A[i],3) == dim :
                for j in range(len(A)):
                    if A[i][j] == 3:
                        A[i][j] = 1
        return C

    def modifyrow(A,ndx):
        C = [[A[i][j] for j in range(len(A))] for i in range(len(A))]
        for i in range(len(A)):
            if i == ndx:
                continue
            if C[ndx][i] == 1 :
                C[i] = compare(C[ndx],C[i],int(len(A)**0.5))
            else:
                C[i] = compare(C[ndx],C[i],2)
            for j in range(len(A)):
                C[j][i] = C[i][j]
        return C

    def pcheck(C):
        m = 0
        p = 0
        i = 0
        j = 0
        l = 0
        lens = len(C)
        dim = 2*int(lens**0.5)-1
        sqr = int(lens**0.5)
        for k in range(lens):
            if count(A2(C,C)[k],dim) == 1:
                m = m + 1
            if count(A2(C,C)[k],sqr) == dim-1:
                p = p + 1
            if count(A2(C,C)[k],2) == (sqr-1)**2:
                i = i + 1
            if pcheckII(C,k) == True:
                j = j + 1
            if pcheckIII(C,k) == True:
                l = l + 1
        if m == lens and p == lens and i == lens
          and j == lens and l == lens:
            return True
        else:
            return False
```

```
def compare3(A,B,C):
    n = 0
    for i in range(len(A)):
        n += A[i]*B[i]*C[i]
    return n

def pcheckII(C,i):
    A = True
    j = 0
    k = 0
    for x in countn(C[i],1):
        if x != i and count(countcompare(C[i],C[x]),1) == 3 :
            j += 1
    for x in countn(C[i],0):
        if count(countcompare(C[i],C[x]),1) == 2 :
            k += 1
    if j == count(C[i],1)-1 and k == count(C[i],0):
        s = countn(C[i],1)[1]
        t = countn(countcompare(C[i],C[s]),1)[2]
        if compare3(C[i],C[s],C[t]) == 3:
            return True
        else:
            return False
    else:
        return False

def pcheckIII(C,i):
    for x in countn(C[i],0):
        if count(countcompare(C[i],C[x]),1) != 2:
            return False
            break
        else:
            return True
            continue

def SolveRtn(A,dim):
    C = [[A[i][j] for j in range(len(A))] for i in range(len(A))]
    info = max3(C)
    if info[1] == 0:
        if pcheck(C):
            printmatrix(ptomap(C))
            for x in C:
                print(x)
            print("..........")
        return 0
    possicom = combinations(dim, info[1], info[2])
    for x in possicom:
        C1 = rowfill(C, info[0], x)
        C2 = modifyrow(C1,info[0])
        C3 = modify3(C2)
        if pstarcheck(C3):
            SolveRtn(C3,dim)
        else:
            continue

def number_to_lattice(n,k):
    a = math.ceil(n/k)
    b = n%k
    if b == 0:
        b = b + k
        return((a,b))
    else:
        return((a,b))

def cardmatrix(n):
    C = [[0 for x in range(n)]for y in range(n)]
    for i in range(n):
        for j in range(n):
            if number_to_lattice(i+1,int(n**0.5))[0] ==
number_to_lattice(j+1,int(n**0.5))[0] or
number_to_lattice(i+1,int(n**0.5))[1] ==
number_to_lattice(j+1,int(n**0.5))[1]:
                C[i][j] = 1
            C[i][i] = 1
    return(C)

def printmatrix(C):
    for x in C:
        print(x)

def removelist(A,B):
    C = []
    for x in A:
        if x not in B:
            C += [x]
        else:
            continue
    return C

def ptomap(P):
    q = int(len(P)**0.5)
    M = [[0 for i in range(q)]for j in range(q)]
    A = countn(P[0],1)
    B = removelist(countn(countcompare(P[0],P[A[1]]),1),[0])
    D = removelist(A,(B+[0]))
    M[0][0] = (1,1)
    for i in range(q-1):
        M[0][i+1] = number_to_lattice(B[i]+1,q)
        M[i+1][0] = number_to_lattice(D[i]+1,q)
    for j in range(q-1):
        for k in range(q-1):
            M[k+1][j+1] =
number_to_lattice(countn(countcompare(P[B[j]],P[D[k]]),1)[1]+1,q)
    return M

def adjmatrix_to_pstar(A):
    dim = len(A)
    C = [[0 for x in range(dim)]for y in range(dim)]
    for i in range(dim):
        for j in range(dim):
            if cardmatrix(dim)[i][j] != 0 :
                C[i][j] = int((A[i][j])/(cardmatrix(dim)[i][j]))
            else:
                C[i][j] = 3
    for k in range(dim):
        C[k][k] = 1
    return(C)

def adjmatrix_to_map(A):
    P = adjmatrix_to_pstar(A)
    SolveRtn(P,5)
```

執行結果

```
>>> A = [
[0, 1, 0, 0, 0, 0, 1, 0, 0],
[1, 0, 0, 0, 1, 0, 0, 1, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 1],
[0, 0, 0, 0, 1, 0, 1, 0, 0],
[0, 1, 0, 1, 0, 1, 0, 0, 0],
[0, 0, 0, 0, 1, 0, 0, 0, 1],
[1, 0, 0, 1, 0, 0, 0, 0, 0],
[0, 1, 0, 0, 0, 0, 0, 0, 1],
[0, 0, 1, 0, 0, 1, 0, 1, 0]]
>>> adjmatrix_to_map(A)
[(1, 1), (1, 2), (3, 2)]
[(2, 3), (2, 2), (3, 3)]
[(3, 1), (2, 1), (1, 3)]
[1, 1, 0, 0, 0, 1, 1, 1, 0]
[1, 1, 0, 1, 1, 0, 0, 1, 0]
[0, 0, 1, 1, 0, 0, 1, 1, 1]
[0, 1, 1, 1, 1, 0, 1, 0, 0]
[0, 1, 0, 1, 1, 1, 0, 0, 1]
[1, 0, 0, 0, 1, 1, 1, 0, 1]
[1, 0, 1, 1, 0, 1, 1, 0, 0]
[1, 1, 1, 0, 0, 0, 0, 1, 1]
[0, 0, 1, 0, 1, 1, 0, 1, 1]
.....
```

【評語】050416

本科展作品探討一遊戲的問題，並將之化為圖論問題，以求圖中兩點之間的最短路徑。文章中給出三種求證，基本上可行，只是時間複雜度不是最好的。另本作品除給予數學方法證明外，另設計程式作驗證，內容尚可。建議本作品除了利用 python 程式外，應找到其數學上存在性的結果。

作品海報

研究目的與定位

【目的】 本研究從圖論的角度探討桌遊「Micro Robots」所引出的相關問題。

一、提出三種方法找任兩點的最短距離、最短路徑與最短路徑數，並給出證明與寫成 Python 程式。

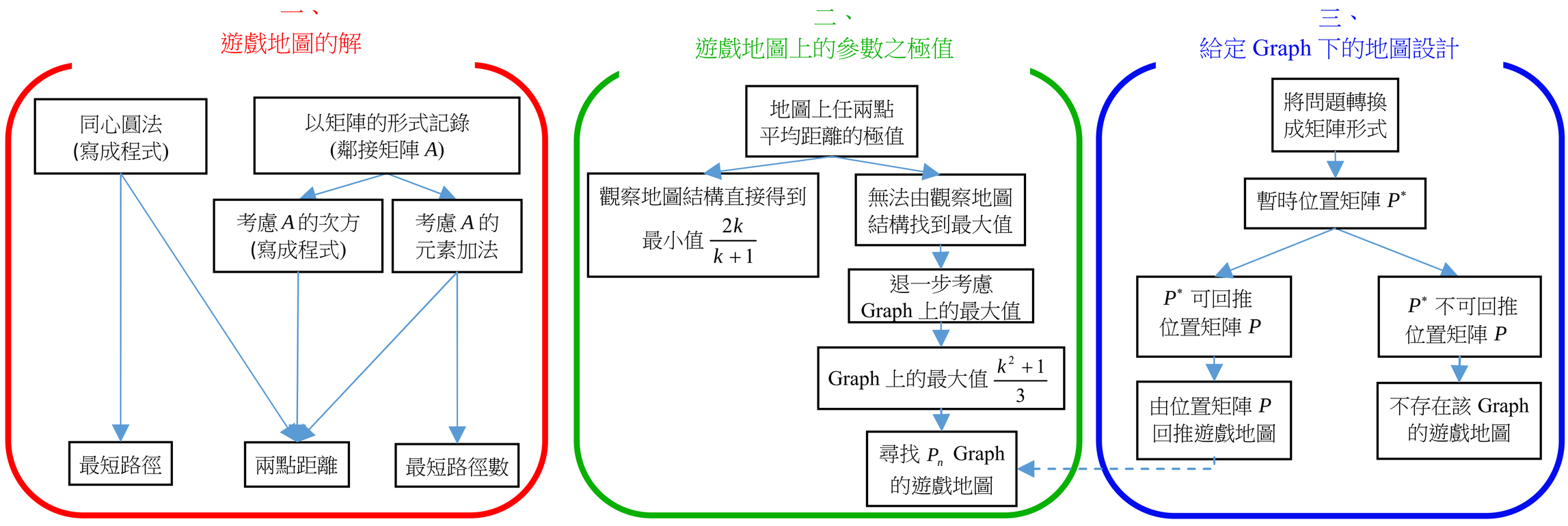
二、探討地圖任兩點平均距離的極值。

三、提出滿足特殊條件的地圖之設計方法，並寫成 Python 程式，最後將其應用在地圖的極端情形上(平均距離的極值)。

【定位】

我們搜尋並參閱科展資料庫後現沒有相關研究，歷屆科展的棋盤問題都是在處理「在固定的 Graph 上找 Hamiltonian path 、Hamiltonian cycle」。而本研究的遊戲地圖與 Graph 是可變動的。除了最短路徑，也研究在給定 Graph 下反找遊戲地圖的問題。研究的第三部分與歷屆作品相較，我們希望設計遊戲地圖，使其 Graph 只能是 Hamiltonian path。另外，本研究提出的演算法適用任何 Graph，只要有 Graph，皆能判別對應的遊戲地圖之存在性。

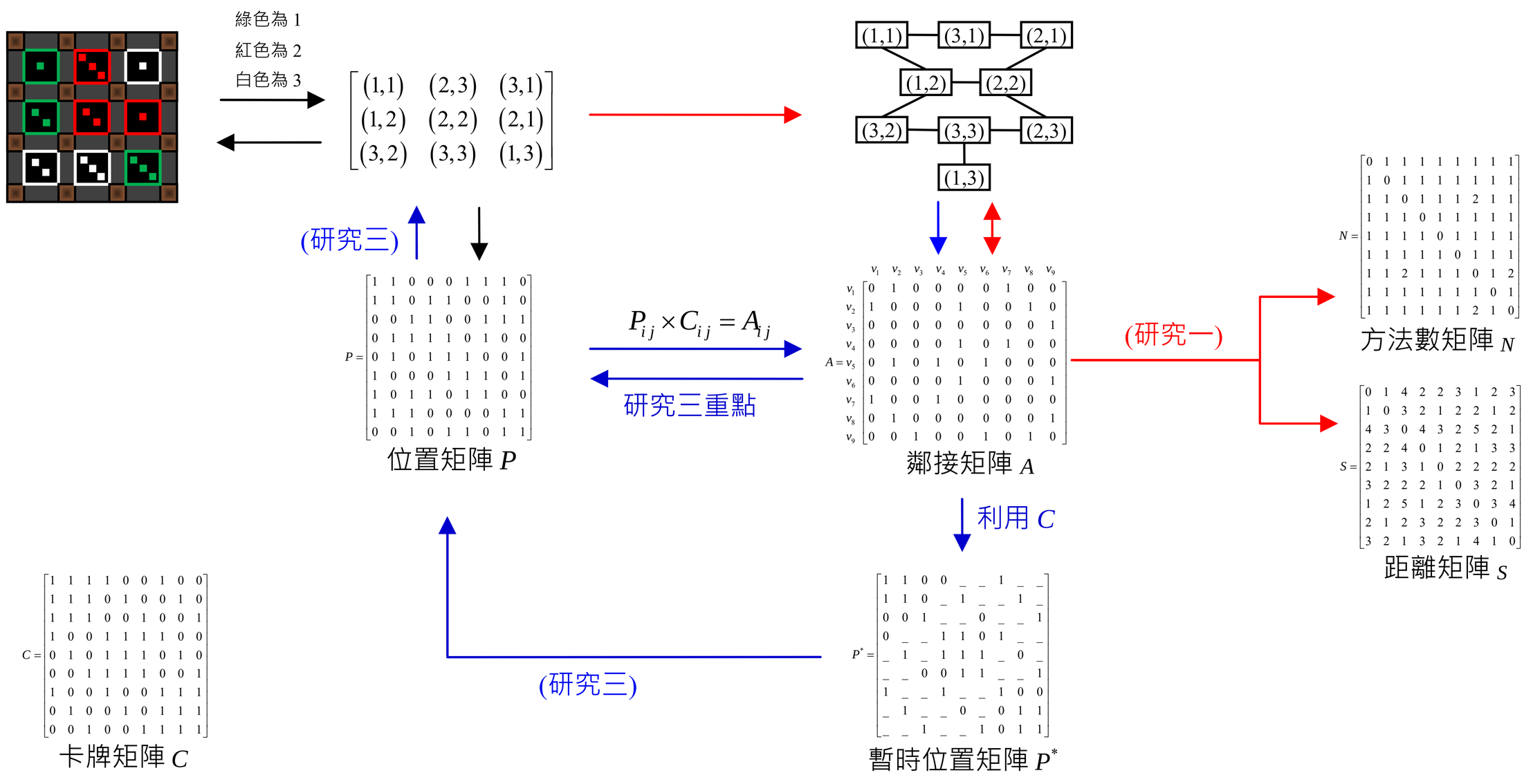
研究架構圖



給定遊戲地圖下，我們可以透過(研究一)找到解(最短路徑、兩點距離、最短路徑數)，
若希望設計滿足特殊條件的地圖(研究二)，我們可以先考慮 Graph 的設計，進而透過(研究三)找到滿足的地圖。

表徵轉換

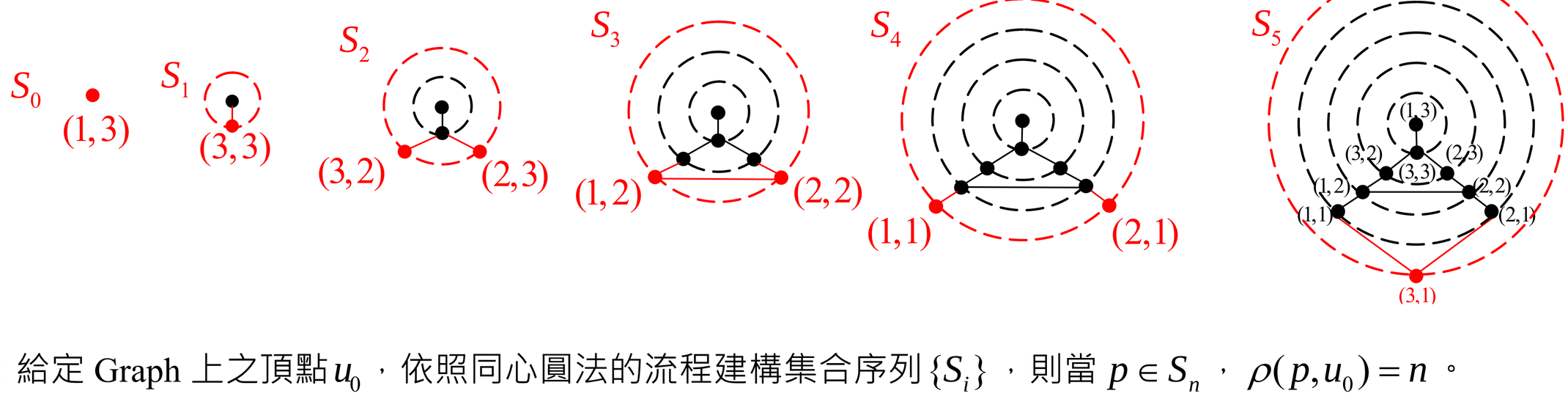
考慮頂點集合 $\{(p,q): p=1,2,\cdots k, q=1,2,\cdots k\}$ ，並將頂點依序標籤如下： $(p,q)=v_{k(p-1)+q}$



研究結果

一、提出三種方法找出最短距離、最短路徑與最短路徑數，並寫成 Python 程式

(1)同心圓法



定理 1：給定 Graph 上之頂點 u_0，依照同心圓法的流程建構集合序列 $\{S_i\}$，則當 $p \in S_n$，$\rho(p,u_0)=n$。

本定理可由數學歸納法得證，詳見說明書。

(2)鄰接矩陣的元素加法

$A_1=\begin{bmatrix}0&1&0&0&0&0&1&0&0\\1&0&0&0&1&0&0&1&0\\0&0&0&0&0&0&0&0&1\\0&0&0&0&1&0&1&0&0\\0&1&0&1&0&1&0&0&0\\0&0&0&0&1&0&0&0&1\\1&0&0&1&0&0&0&0&0\\0&1&0&0&0&0&0&0&1\\0&0&1&0&0&1&0&1&0\end{bmatrix}$ $A_2=\begin{bmatrix}0&1&0&2&2&0&1&2&0\\1&0&0&2&1&2&2&1&2\\0&0&0&0&0&2&0&2&1\\2&2&0&0&1&2&1&0&0\\2&1&0&1&0&1&2&2&2\\0&2&2&2&1&0&0&2&1\\1&2&0&1&2&0&0&0&0\\2&1&2&0&2&2&0&0&1\\0&2&1&0&2&1&0&1&0\end{bmatrix}$ $A_3=\begin{bmatrix}0&1&0&2&2&3&1&2&3\\1&0&3&2&1&2&2&1&2\\0&3&0&0&3&2&0&2&1\\2&2&0&0&1&2&1&3&3\\2&1&3&1&0&1&2&2&2\\3&2&2&2&1&0&3&2&1\\1&2&0&1&2&3&0&3&0\\2&1&2&3&2&2&3&0&1\\3&2&1&3&2&1&0&1&0\end{bmatrix}$ $A_4=\begin{bmatrix}0&1&4&2&2&3&1&2&3\\1&0&3&2&1&2&2&1&2\\4&3&0&4&3&2&0&2&1\\2&2&4&0&1&2&1&3&3\\2&1&3&1&0&1&2&2&2\\3&2&2&2&1&0&3&2&1\\1&2&0&1&2&3&0&3&4\\2&1&2&3&2&2&3&0&1\\3&2&1&3&2&1&4&1&0\end{bmatrix}$ $A_5=\begin{bmatrix}0&1&4&2&2&3&1&2&3\\1&0&3&2&1&2&2&1&2\\4&3&0&4&3&2&5&2&1\\2&2&4&0&1&2&1&3&3\\2&1&3&1&0&1&2&2&2\\3&2&2&2&1&0&3&2&1\\1&2&5&1&2&3&0&3&4\\2&1&2&3&2&2&3&0&1\\3&2&1&3&2&1&4&1&0\end{bmatrix}$

$$A_1=A\text{、}(A_{k+1})_{ij}=\begin{cases}0 & if\quad i=j\\ (A_k)_{ij} & if\quad 0<(A_k)_{ij}<k+1\ \wedge i\neq j\\ k+1 & if\quad (A_k)_{ij}=0\ \wedge\ \exists\, p\ni:(A_k)_{ip}+(A_k)_{pj}=k+1\ \wedge\ i\neq j\\ 0 & if\quad \nexists\, p\ni:(A_k)_{ip}+(A_k)_{pj}=k+1\wedge i\neq j\end{cases},\ \forall k\geq 1$$

定理 2：若 Graph 為連通圖且 A_m 滿足除了主對角線外其餘不為 0，則 A_m 即為距離矩陣 S。同樣可由數學歸納法得證。

(3)鄰接矩陣的乘法

$A^1=\begin{bmatrix}0&1&0&0&0&0&1&0&0\\1&0&0&0&1&0&0&1&0\\0&0&0&0&0&0&0&0&1\\0&0&0&0&1&0&1&0&0\\0&1&0&1&0&1&0&0&0\\0&0&0&0&1&0&0&0&1\\1&0&0&1&0&0&0&0&0\\0&1&0&0&0&0&0&0&1\\0&0&1&0&0&1&0&1&0\end{bmatrix}$ $A^2=\begin{bmatrix}2&0&0&1&1&0&0&1&0\\0&3&0&1&0&1&1&0&1\\0&0&1&0&0&1&0&1&0\\1&1&0&2&0&1&0&0&0\\1&0&0&0&3&0&1&1&1\\0&1&1&1&0&2&0&1&0\\0&1&0&0&1&0&2&0&0\\1&0&1&0&1&0&0&2&0\\0&1&0&0&1&0&0&0&3\end{bmatrix}$ $A^3=\begin{bmatrix}0&4&0&1&1&1&3&0&1\\4&0&1&1&5&1&1&4&1\\0&1&0&0&1&0&0&0&3\\1&1&0&0&4&0&3&1&1\\1&5&1&4&0&4&1&1&1\\1&1&0&0&4&0&1&1&4\\3&1&0&3&1&1&0&1&0\\0&4&0&1&1&1&1&0&4\\1&1&3&1&1&4&0&4&0\end{bmatrix}$ $A^4=\begin{bmatrix}7&1&1&4&6&2&1&5&1\\1&13&1&6&2&6&5&1&6\\1&1&3&1&1&4&0&4&0\\4&6&1&7&1&5&1&2&1\\6&2&1&1&13&1&5&6&6\\2&6&4&5&1&8&1&5&1\\1&5&0&1&5&1&6&1&2\\5&1&4&2&6&5&1&8&1\\1&6&0&1&6&1&2&1&11\end{bmatrix}$ $A^5=\begin{bmatrix}2&18&1&7&7&7&11&2&8\\18&4&6&7&25&8&7&19&8\\1&6&0&1&6&1&2&1&11\\7&7&1&2&18&2&11&7&8\\7&25&6&18&4&19&7&8&8\\7&8&1&2&19&2&7&7&17\\11&7&2&11&7&7&2&7&2\\2&19&1&7&8&7&7&2&17\\8&8&11&8&8&17&2&17&2\end{bmatrix}$ $N=\begin{bmatrix}0&1&1&1&1&1&1&1&1\\1&0&1&1&1&1&1&1&1\\1&1&0&1&1&1&2&1&1\\1&1&1&0&1&1&1&1&1\\1&1&1&1&0&1&1&1&1\\1&1&1&1&1&0&1&1&1\\1&1&2&1&1&1&0&1&2\\1&1&1&1&1&1&1&0&1\\1&1&1&1&1&1&2&1&0\end{bmatrix}$ $S=\begin{bmatrix}0&1&4&2&2&3&1&2&3\\1&0&3&2&1&2&2&1&2\\4&3&0&4&3&2&5&2&1\\2&2&4&0&1&2&1&3&3\\2&1&3&1&0&1&2&2&2\\3&2&2&2&1&0&3&2&1\\1&2&5&1&2&3&0&3&4\\2&1&2&3&2&2&3&0&1\\3&2&1&3&2&1&4&1&0\end{bmatrix}$

定理 3：v_i 到 v_j 走 p 步到達的方法數有 $(A^p)_{ij}$ 種。

證明：$(A^p)_{ij}=\sum_{n_1=1}^{n}(A^{p-1})_{in_1}A_{n_1j}=\cdots=\sum_{n_1=1}^{n}\cdots\sum_{n_{p-1}=1}^{n}A_{in_{p-1}}A_{n_{p-1}n_{p-2}}\cdots A_{n_1j}$，若 $(A^p)_{ij}=r$，由於 $A_{ij}=1\vee 0$，這表示 $A_{in_{p-1}}A_{n_{p-1}n_{p-2}}\cdots A_{n_1j}=1$ 有 r 組，

即 $(v_i,v_{n_{p-1}}),(v_{n_{p-1}},v_{n_{p-2}})\cdots(v_{n_1},v_j)\in E(G)$ 有 r 組，故 $(A^p)_{ij}$ 表示 v_i 到 v_j 走 p 步到達的方法數有 r 種。

一般化作法：$S_{ij}=\min\{p\,|\,(A^p)_{ij}\neq 0\}$，$N_{ij}=(A^{S_{ij}})_{ij}$。

我們分別將同心圓法、鄰接矩陣元素的加法、鄰接矩陣的乘法寫成 Python 程式，詳見說明書附錄一、二、三。

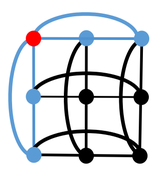
(4)各演算法的比較

演算法	同心圓法	鄰接矩陣的元素加法	鄰接矩陣的乘法
功能	任兩點最短距離、最短路徑	任兩點最短距離	任兩點最短距離、最短路徑數
時間複雜度	$O(k^7)$	$O(k^8)$	$O(k^{14})$

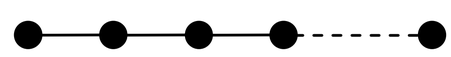
二、探討地圖任兩點平均距離的極值

(1) 平均距離的最小值

$\begin{bmatrix}(1,1)&(1,2)&(1,3)\\(2,1)&(2,2)&(2,3)\\(3,1)&(3,2)&(3,3)\end{bmatrix}$



(2) 平均距離的最大值



定理 4：遊戲地圖的平均距離有最小值 $\frac{2k}{k+1}$

定理 5：若 Graph 為 P_n，則相異兩點平均距離有最大值 $\frac{(n+1)}{3}$

三、提出滿足特殊條件的地圖之設計方法，並將其應用在地圖的極端情形(平均距離的極值)上

(1)「位置矩陣 P」、「卡牌矩陣 C」、「暫時位置矩陣 P^*」

卡牌矩陣 C：若 v_i、v_j 的第 1 個元相同或第 2 個元相同，則 $C_{ij}=1$，否則 $C_{ij}=0$。

位置矩陣 P：若 v_i、v_j 的地圖位置在同一行或同一列，則 $P_{ij}=1$，否則 $P_{ij}=0$。

} 因此當 $i\neq j$ 時，$P_{ij}\times C_{ij}=A_{ij}$

定義暫時位置矩陣 P^*：當 $i=j$，$P^*_{ij}=1$。當 $i\neq j$，若 $C_{ij}\neq 0$，則 $P^*_{ij}=\frac{A_{ij}}{C_{ij}}$，否則 $P^*_{ij}=\_$(待定數值)

(2) 在 $k\times k$ 的「地圖」中，「位置矩陣 P」的性質

性質 1：P 為對稱矩陣。

$P=\begin{bmatrix}1&1&0&0&0&1&1&1&0\\1&1&0&1&1&0&0&1&0\\0&0&1&1&0&0&1&1&1\\0&1&1&1&1&0&1&0&0\\0&1&0&1&1&1&0&0&1\\1&0&0&0&1&1&1&0&1\\1&0&1&1&0&1&1&0&0\\1&1&1&0&0&0&0&1&1\\0&0&1&0&1&1&0&1&1\end{bmatrix}$

性質 2：P 的每一行的元中，必有 $2k-1$ 個 1，$(k-1)^2$ 個 0。

$\begin{bmatrix}(1,1)&(2,3)&(3,1)\\(1,2)&(2,2)&(2,1)\\(3,2)&(3,3)&(1,3)\end{bmatrix}$

性質 3：若 $P_{ij}=1$，則 P 的第 i 行與第 j 行恰好在相同的 k 個列中皆為 1 即 $\left|\{m\,|\,P_{mi}=P_{mj}=1\}\right|=k$。

$P=\begin{bmatrix}\mathbf{1}&1&0&0&0&1&\mathbf{1}&1&0\\1&1&0&1&1&0&0&1&0\\0&0&1&1&0&0&1&1&1\\0&1&1&1&1&0&1&0&0\\0&1&0&1&1&1&0&0&1\\\mathbf{1}&0&0&0&1&1&\mathbf{1}&0&1\\\mathbf{1}&0&1&1&0&1&\mathbf{1}&0&0\\1&1&1&0&0&0&0&1&1\\0&0&1&0&1&1&0&1&1\end{bmatrix}$

性質 4：若 $P_{ij}=0$，則 P 的第 i 行與第 j 行恰好在相同的 2 個列中皆為 1，即 $\left|\{m\,|\,P_{mi}=P_{mj}=1\}\right|=2$。

$P=\begin{bmatrix}1&1&0&0&0&1&1&1&0\\\mathbf{1}&1&0&\mathbf{1}&1&0&0&1&0\\0&0&1&1&0&0&1&1&1\\0&1&1&1&1&0&1&0&0\\0&1&0&1&1&1&0&0&1\\\mathbf{1}&0&0&0&\mathbf{1}&1&1&0&1\\1&0&1&1&0&1&1&0&0\\1&1&1&0&0&0&0&1&1\\0&0&1&0&1&1&0&1&1\end{bmatrix}$

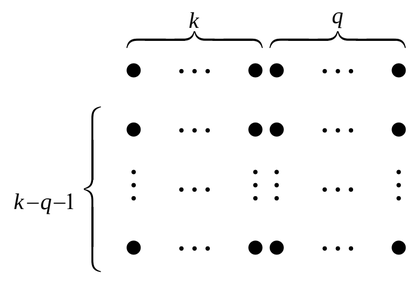
定理 6：若 $Q_{k^2\times k^2}$ 滿足性質 1~4，則存在「遊戲地圖」使其該地圖的「位置矩陣」為 $Q_{k^2\times k^2}$。

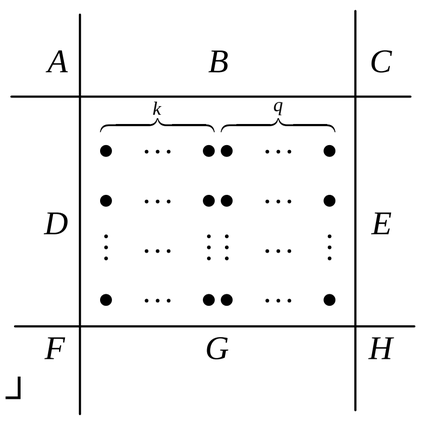
證明：目標是找到「遊戲地圖」，因為 Q 是 $k^2\times k^2$ 的矩陣，我們可以把 Q_{ij} 想成頂點 v_i 與 v_j 的關係，即 $Q_{ij}=1$ 代表 v_i 與 v_j 要在同一行或同一列，而 $Q_{ij}=0$ 代表 v_i 與 v_j 不在同一行且不在同一列。先將 v_1 置於棋盤 $(1,1)$ 位置，考慮第1行，根據性質 2 可以找到 $2(k-1)$ 個頂點與之同行或同列，再由性質 3，我們可從中區分成 $k-1$ 個頂點與之同行，將這 $k-1$ 個頂點隨機置入棋盤的第一行，不妨假設為 $v_{g(2)},v_{g(3)},\cdots,v_{g(k)}$。另外 $k-1$ 個頂點與之同列，這 $k-1$ 個頂點隨機置入棋盤的第一列，不妨假設為 $v_{f(2)},v_{f(3)},\cdots,v_{f(k)}$。

考慮棋盤 (i,j) 位置 $i>1$、$j>1$，因為 $v_{g(i)}$ 與 $v_{f(j)}$ 不在同一行，也不在同一列，根據性質 4，可找到 2 個頂點與之同行或同列，當然其中一個是 v_1，另一個則填在棋盤 (i,j) 位置。

v_1	$v_{f(2)}$	$v_{f(3)}$	…	$v_{f(k)}$
$v_{g(2)}$				
$v_{g(3)}$				
$v_{g(k)}$				

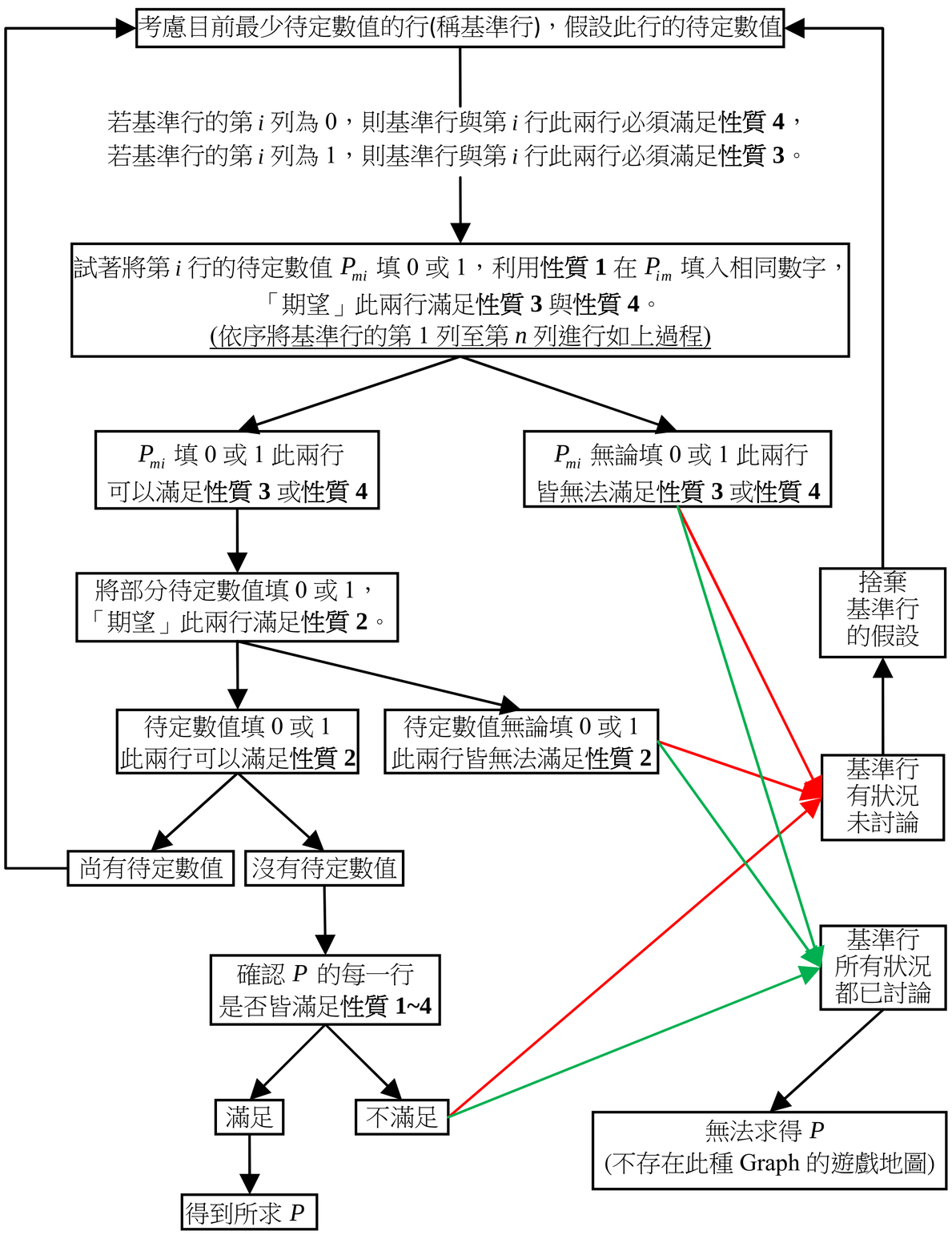
事實上，在$k^2\times k^2$的棋盤上若第i行第j列有頂點，第i行(j列)必恰有k個頂點，即所有頂點形成$k\times k$的方陣。這是因為，若棋盤第j列的頂點數為p且$p>k$，不妨假設$p=k+q(0<q<k-1)$，根據**性質 2**，第i行必須再補足$k-q-1$個點，同理第j列上有頂點的行也都要補足$k-q-1$個點。因此我們會用掉$(k+q)(k-q-1+1)=k^2-q^2$個頂點。





因為$q>0$，所剩的點$q^2>0$，剩下的點必不能放在區域B,D,G,E，否則會違背**性質 2**，但在區域A,C,F,H，與**性質 4**矛盾，故$q=0$，得證$p=k$，即所有頂點的位置會形成$k\times k$的方陣。**定理 6** 除了告訴我們滿足性質 1~4 的$Q_{k^2\times k^2}$是某「遊戲地圖」的「位置矩陣P」，其證明過程亦提供了一套由「位置矩陣P」回到「遊戲地圖」的方法。

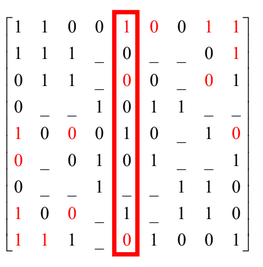
(3)「暫時位置矩陣 P^*」反推「位置矩陣 P」



例子：

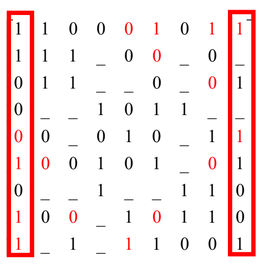
$$P^*=\begin{bmatrix}1&1&0&0&\_&\_&0&\_&\_\\1&1&1&\_&0&\_&\_&0&\_\\0&1&1&\_&\_&0&\_&\_&1\\0&\_&\_&1&0&1&1&\_&\_\\\_&0&\_&0&1&0&\_&1&\_\\\_&\_&0&1&0&1&\_&\_&1\\0&\_&\_&1&\_&\_&1&1&0\\\_&0&\_&\_&1&\_&1&1&0\\\_&\_&1&\_&\_&1&0&0&1\end{bmatrix}$$

從基準行(第 1 行)開始將底線依序假設為 1,0,1,1



依據性質 2，產生矛盾。

從基準行(第 1 行)開始將底線依序假設為 0,1,1,1

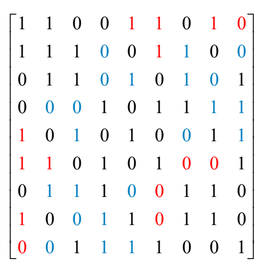


依據性質 3，產生矛盾。

從基準行(第 1 行)開始將底線依序假設為 1,1,0,1

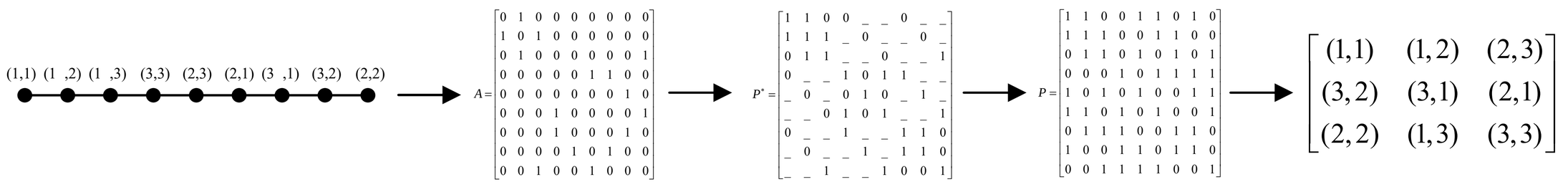
依據性質 3，產生矛盾。

從基準行(第 1 行)開始將底線依序假設為 1,1,1,0



此矩陣滿足 P 的性質，故為解。

我們利用(1)(2)(3)提出的地圖設計方法，



給定 Graph　　鄰接矩陣　　暫時位置矩陣　　位置矩陣　　遊戲地圖

一般來說，我們可以考慮「遊戲地圖」上的各種參數(本研究著重在平均距離)，而當我們計算這些參數的極值，某些時候在 Graph 上會比在「遊戲地圖」上簡單，而且有許多 Graph 理論提供分析，因此我們可以考慮 Graph 上的參數，再想辦法找到「遊戲地圖」來對應 Graph。

同理，要設計 Graph 為P_{k^2}的$k\times k$「遊戲地圖」，我們可以考慮各種連接順序的P_{k^2} Graph，利用本研究的方法得到「遊戲地圖」。

研究結論

一、給定遊戲地圖

(1) 將 Graph 轉為同心圓，可以從同心圓上讀出距離與路徑。

(2) 利用「鄰接矩陣A」的特殊加法，可以計算任兩點的距離。

(3) 利用「鄰接矩陣A」的乘法，可以計算任兩點的距離，並求得最短距離下的方法數。

二、地圖平均距離的極值

(1) 遊戲地圖的(i,j)位置放點(i,j)，有平均距離最小值$\frac{2k}{k+1}$。

(2) 在n點的 Graph 下，其平均距離的最大值為$\frac{n+1}{3}$，且 Graph 必為P_n。

三、探討給定 Graph 下，遊戲地圖設計的方法與可行性

(1) 將遊戲地圖設計的問題轉換成矩陣形式。

(2) 遊戲地圖的位置矩陣有四個性質，反之滿足這四個性質的矩陣，也是某遊戲地圖的位置矩陣。

(3) 給出一套從「暫時位置矩陣P^*」推得「位置矩陣P」的過程，並推得原始遊戲地圖。