數獨遊戲 (Sudoku)

丁培毅

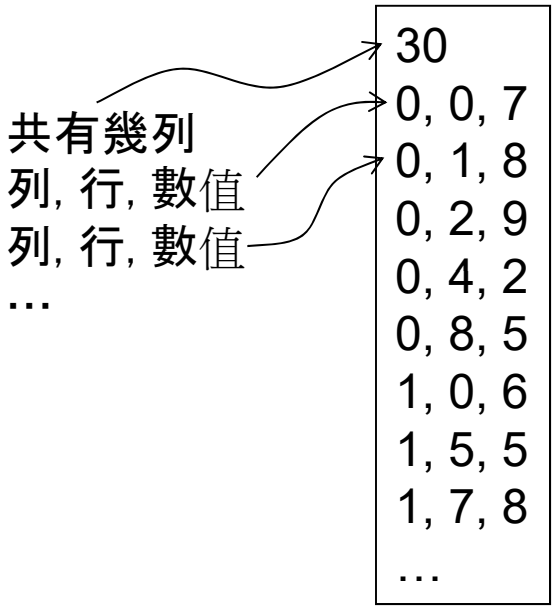
實習目標：

1. 練習運用迴圈/遞迴產生各種排列的演算法
2. 運用函式簡化演算法的設計
3. 深度優先搜尋法
4. Backtracking 尋找下一組可能的答案
5. 運用 clock() 函式計算程式執行時間

數獨遊戲 (Sudoku)

- Sudoku: 右下三個範例中，81 格可以分為 9 個子區，每一區是一個 3x3 的九宮格。玩家就是要找出每一格可以填進去的數字，使得每一列、每一行、以及每一個九宮格中都不會出現重複的數字 (都是 1,2,3,…,9 的排列)。
- 請寫一個程式來完成填數字的任務，輸入是下圖左的檔案，代表下圖的遊戲，內容說明如下

		6	1		3	4		
		3			8			
5	4		7			1	2	
			2					4
	5		3		9		7	
7					4			
	3	7			5		4	2
			8			7		
		1	4		7	8		



7	8	9		2				5
6					5		8	
							1	
2			8				5	1
		5	7		1	6		
9	1				6			3
	7							
	5		9					6
8				1		5	3	7

3		8				6		
	6					4		
	9		8	5			1	
2			9					
4		5				9		2
					7			6
	2			1	3		6	
		9					3	
		1				5		4

- 範例執行程式之輸出如下

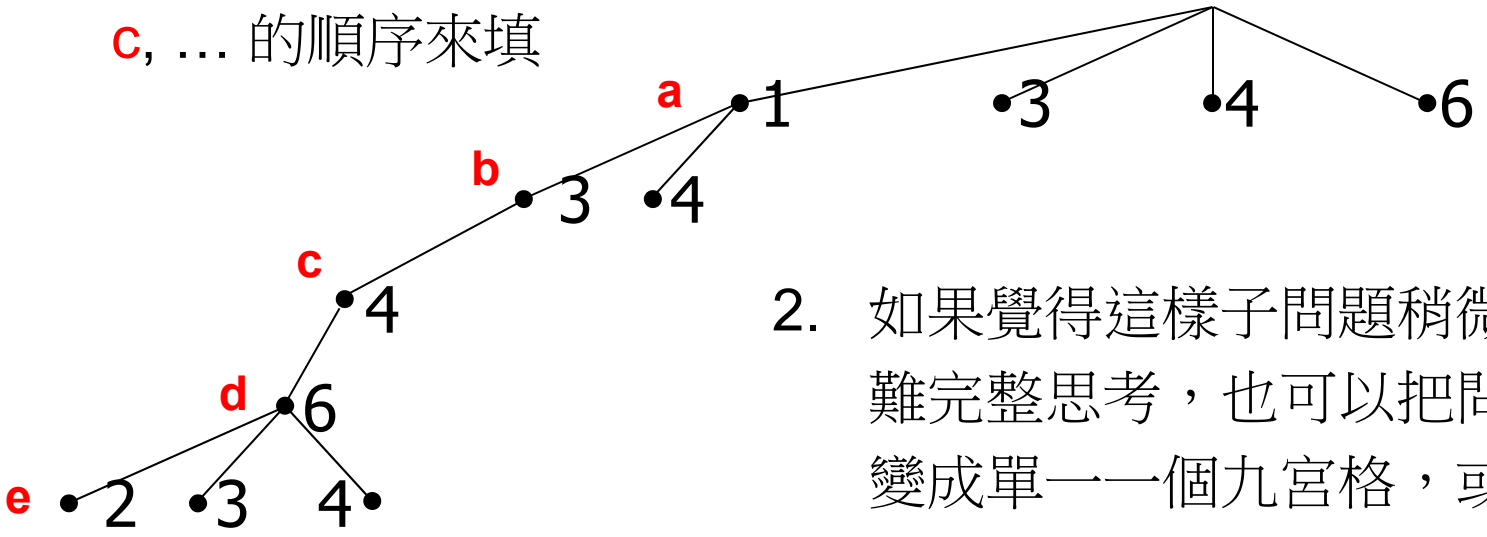
```
Solution:   1
    | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
 ===+===+===+===+===+===+===+===+===+===+
  0 | 7 | 8 | 9 | 1 | 2 | 4 | 3 | 6 | 5 |
 ---+---+---+---+---+---+---+---+---+---+---+
  1 | 6 | 4 | 1 | 3 | 7 | 5 | 2 | 8 | 9 |
 ---+---+---+---+---+---+---+---+---+---+---+
  2 | 5 | 2 | 3 | 6 | 8 | 9 | 7 | 1 | 4 |
 ---+---+---+---+---+---+---+---+---+---+---+
  3 | 2 | 6 | 7 | 8 | 4 | 3 | 9 | 5 | 1 |
 ---+---+---+---+---+---+---+---+---+---+---+
  4 | 4 | 3 | 5 | 7 | 9 | 1 | 6 | 2 | 8 |
 ---+---+---+---+---+---+---+---+---+---+---+
  5 | 9 | 1 | 8 | 2 | 5 | 6 | 4 | 7 | 3 |
 ---+---+---+---+---+---+---+---+---+---+---+
  6 | 3 | 7 | 4 | 5 | 6 | 8 | 1 | 9 | 2 |
 ---+---+---+---+---+---+---+---+---+---+---+
  7 | 1 | 5 | 2 | 9 | 3 | 7 | 8 | 4 | 6 |
 ---+---+---+---+---+---+---+---+---+---+---+
  8 | 8 | 9 | 6 | 4 | 1 | 2 | 5 | 3 | 7 |
 ---+---+---+---+---+---+---+---+---+---+---+

Total 1 solutions found in 0.0 seconds
```

如果有多組答案的話，請輸出總共有幾組，並且輸出程式花在找出所有答案所需要的時間

問題分析

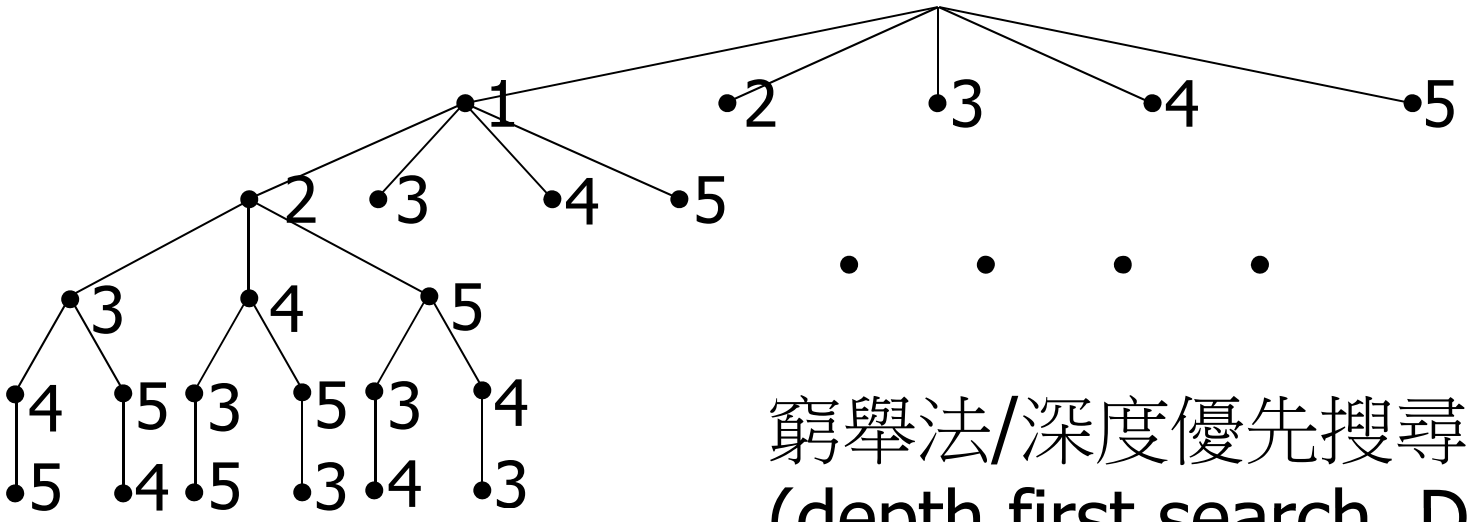
7	8	9	a	2	b	c	d	5
6	e	.	. .		5		8	
							1	
2			8				5	1
		5	7		1	6		
9	1				6			3
	7							
	5		9					6
8				1		5	3	7

1. 要寫程式解這個問題當然先要自己能夠解這個問題，最直接的作法就是一個一個空格逐一嘗試，每一格在填的時候先把和已經填好的數字沒有衝突的數字寫出來，然後再用一樣的方法填下一格，如果填到某一格發現所有可以填的通通和盤面上已經填進去的有衝突，那就回到前一格，用下一個選擇，可以用下面的樹狀決策圖來說明：假設我們依照 a, b, c, … 的順序來填



2. 如果覺得這樣子問題稍微大了一些有點難完整思考，也可以把問題簡化一下，變成單一一個九宮格，或是一條直線，都會變成一個 1~9 排列的問題，只要不和已經選定的數字衝突就好

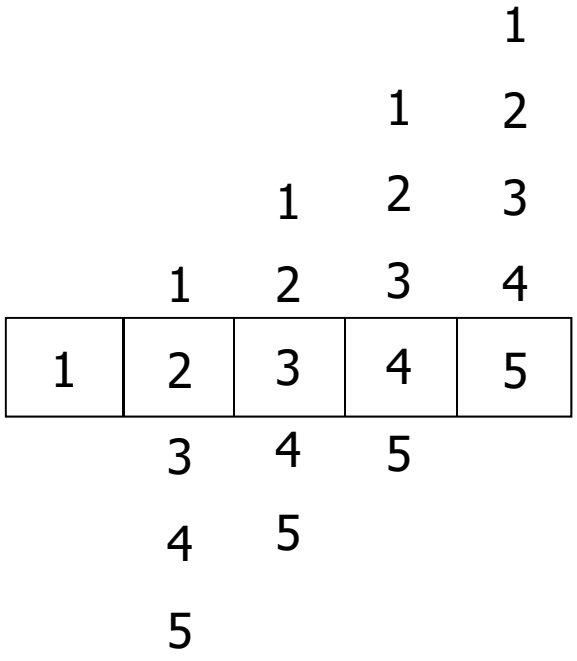
n 個整數的排列問題



窮舉法/深度優先搜尋
(depth first search, DFS)

5! = 120
排列方法

1 2 3 4 5
1 2 3 5 4
1 2 4 3 5
1 2 4 5 3
1 2 5 3 4
1 2 5 4 3
…



兩層 for 迴圈來產生 n! 種排列

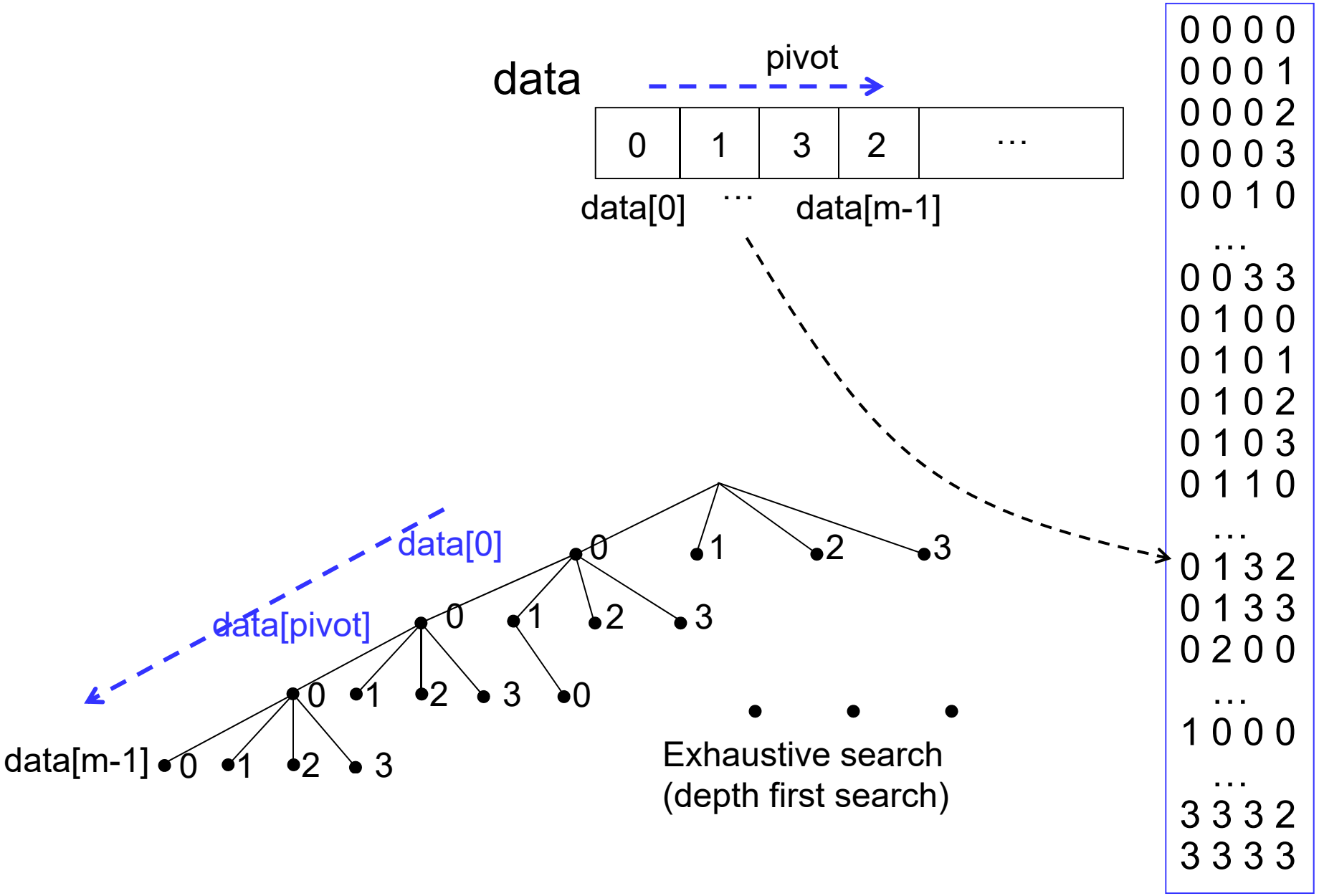
```
int index[4], n=4;
for (i=0; i<n; i++)
    index[i] = i+1;
for (; index[0]<=n; nextIndex(index, n))
    if (isValid(index, n))
        printPerm(index, n);
```

index	1	2	3	4
	1	2	4	3
	1	3	2	4
	1	3	4	2
	1	4	2	3
	1	4	3	2
	2	1	3	4
	2	1	4	3
			...	

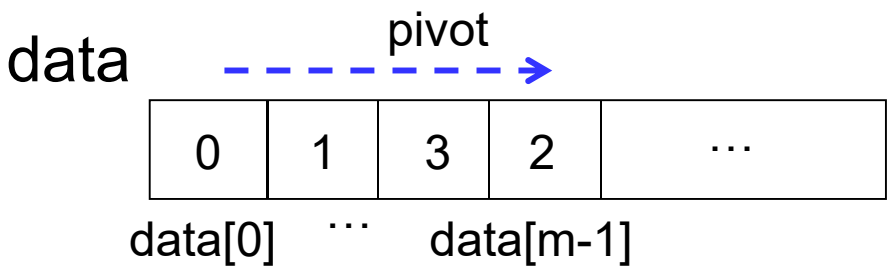
```
int isValid(int index[], int n){
    int i, j;
    for (i=0; i<n-1; i++)
        for (j=i+1; j<n; j++)
            if (index[i]==index[j])
                return 0;
    return 1;
}
```

```
void nextIndex(int index[], int n){
    for (int i=n-1; i>0; i--)
        if (index[i]<n)
            { index[i]++; return; }
        else
            index[i] = 1;
    index[0]++;
}
```

遞迴函式數數字


data
pivot
0
1
3
2
…
data[0]
…
data[m-1]
data[0]
data[pivot]
data[m-1]
Exhaustive search
(depth first search)
0 0 0 0
0 0 0 1
0 0 0 2
0 0 0 3
0 0 1 0
…
0 0 3 3
0 1 0 0
0 1 0 1
0 1 0 2
0 1 0 3
0 1 1 0
…
0 1 3 2
0 1 3 3
0 2 0 0
…
1 0 0 0
…
3 3 3 2
3 3 3 3

遞迴函式數數字




```
01 int main() {
02     int data[10];
03     countUp(4, data, 4, 0);
04     return 0;
05 }
06
07 void countUp(int n, int data[], int m, int pivot) {
08     if (pivot>=m)
09         print(data, m);
10     else
11         for (data[pivot]=0; data[pivot]<n; data[pivot]++)
12             countUp(n, data, m, pivot+1);
13 }
```

base

digits

the running digit

```
0 0 0 0
0 0 0 1
0 0 0 2
0 0 0 3
0 0 1 0
  …
0 0 3 3
0 1 0 0
0 1 0 1
0 1 0 2
0 1 0 3
0 1 1 0
  …
0 1 3 2
0 1 3 3
0 2 0 0
  …
1 0 0 0
  …
3 3 3 2
3 3 3 3
```

資料表示方法

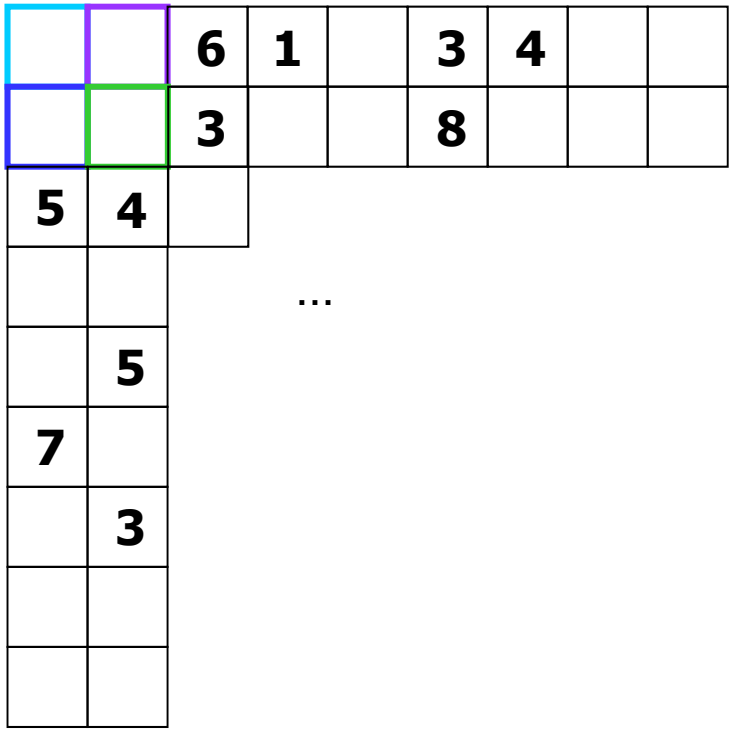
0	0	6	1	0	3	4	0	0
0	0	3	0	0	8	0	0	0
5	4	0	7	0	0	1	2	0
0	0	0	2	0	0	0	0	4
0	5	0	3	0	9	0	7	0
7	0	0	0	0	4	0	0	0
0	3	7	0	0	5	0	4	2
0	0	0	8	0	0	7	0	0
0	0	1	4	0	7	8	0	0

- 二維整數陣列: int board[9][9];
  初始化為 0 以及已知的固定限制

- 前一頁的 index[] 陣列中每一個元素都是要慢慢測試的, 但是上面的 board[9][9] 中有一些是固定的, 你可以把固定的設為負數, 這樣子可以很快跳過這些數, 也可以用一維整數陣列: int position[81]; 或是 int row[81], col[81];
  int nPos=0;
  記錄上面 board[9][9] 陣列中哪些位置是需要填寫的
  例如 0, 1, 4, 7, 8, 9, 10, 12, 13, 15, 16, 17, ….

深度優先搜尋

- 基本的解法延伸前面產生各種排列的演算法，但是可以排進去的數字以及限制都改變了

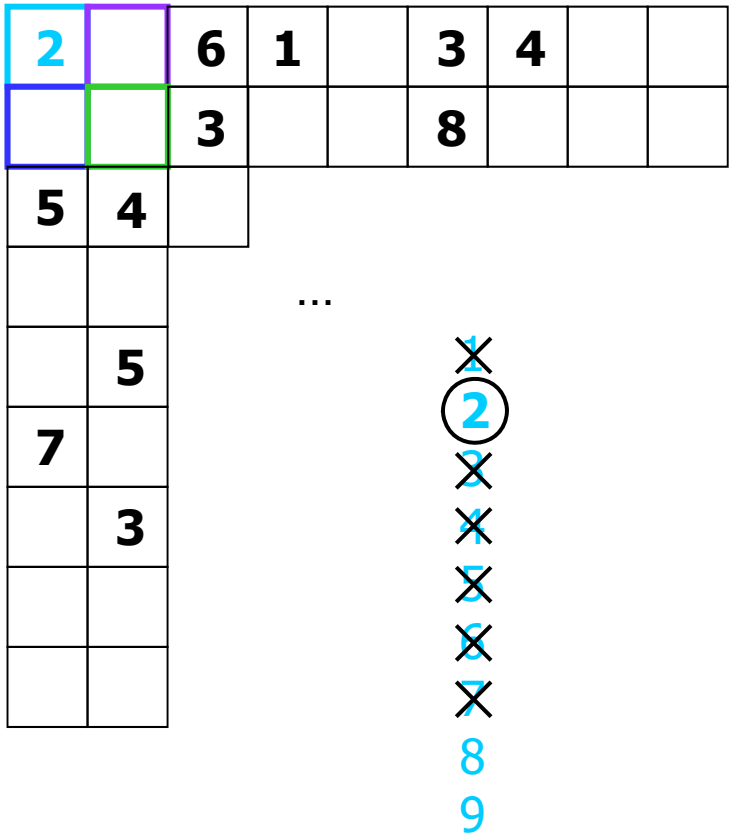


		6	1		3	4		
		3			8			
5	4		7			1	2	
			2					4
7	5		3		9 4		7	
	3	7			5		4	2
			8			7		
		1	4		7	8		

深度優先搜尋

- 基本的解法延伸前面產生各種排列的演算法，但是可以排進去的數字以及限制都改變了

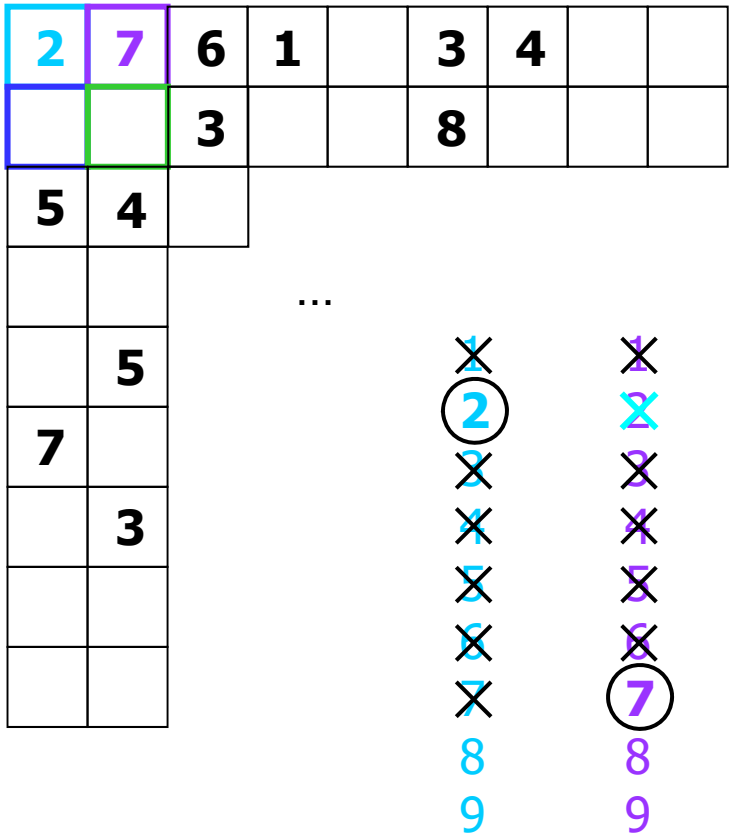
		6	1		3	4		
		3			8			
5	4		7			1	2	
			2					4
7	5		3		9 4		7	
	3	7			5		4	2
			8			7		
		1	4		7	8		



深度優先搜尋

- 基本的解法延伸前面產生各種排列的演算法，但是可以排進去的數字以及限制都改變了

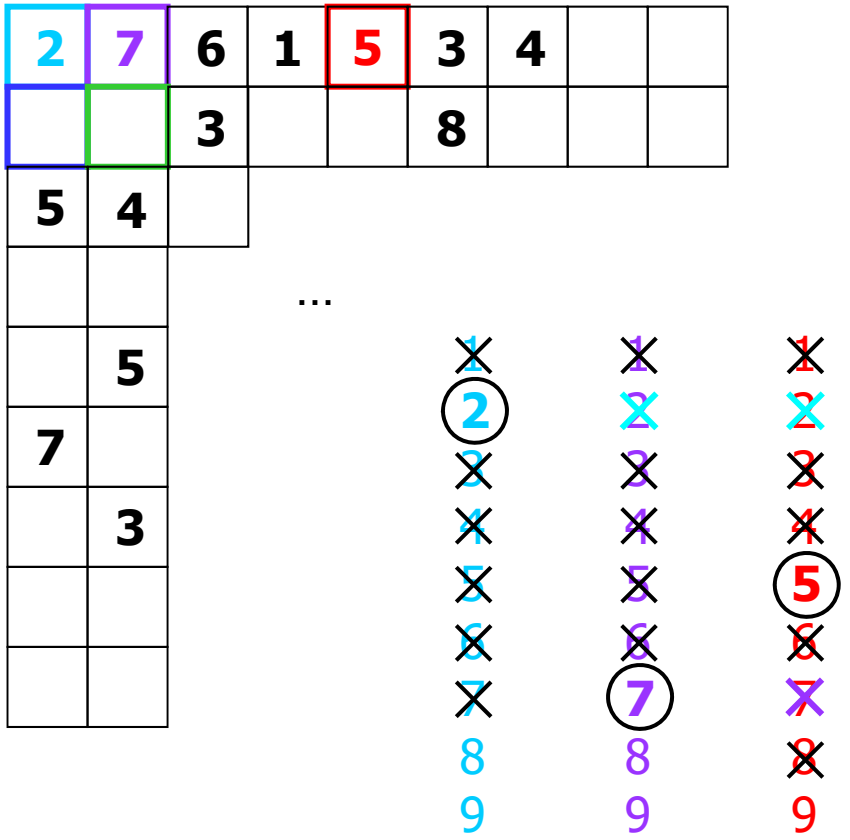
		6	1		3	4		
		3			8			
5	4		7			1	2	
			2					4
7	5		3		9 4		7	
	3	7			5		4	2
			8			7		
		1	4		7	8		



深度優先搜尋

- 基本的解法延伸前面產生各種排列的演算法，但是可以排進去的數字以及限制都改變了

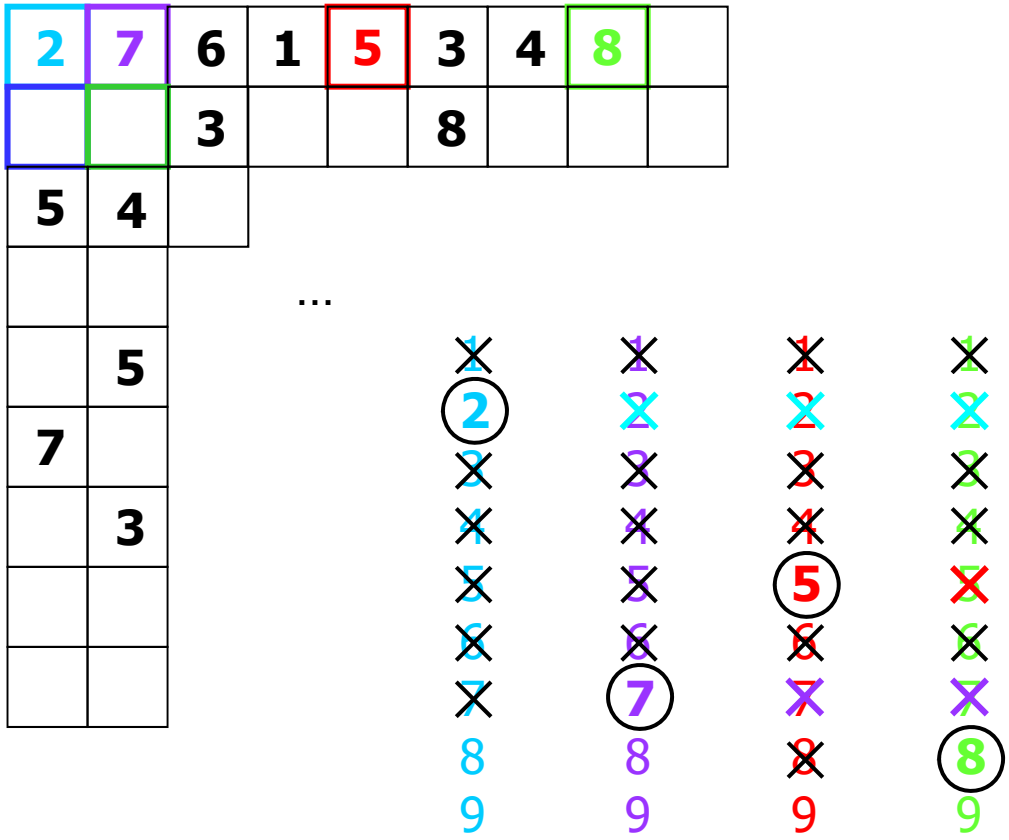
		6	1		3	4		
		3			8			
5	4		7			1	2	
			2					4
7	5		3		9 4		7	
	3	7			5		4	2
			8			7		
		1	4		7	8		



深度優先搜尋

- 基本的解法延伸前面產生各種排列的演算法，但是可以排進去的數字以及限制都改變了

		6	1		3	4		
		3			8			
5	4		7			1	2	
			2					4
7	5		3		9 4		7	
	3	7			5		4	2
			8			7		
		1	4		7	8		



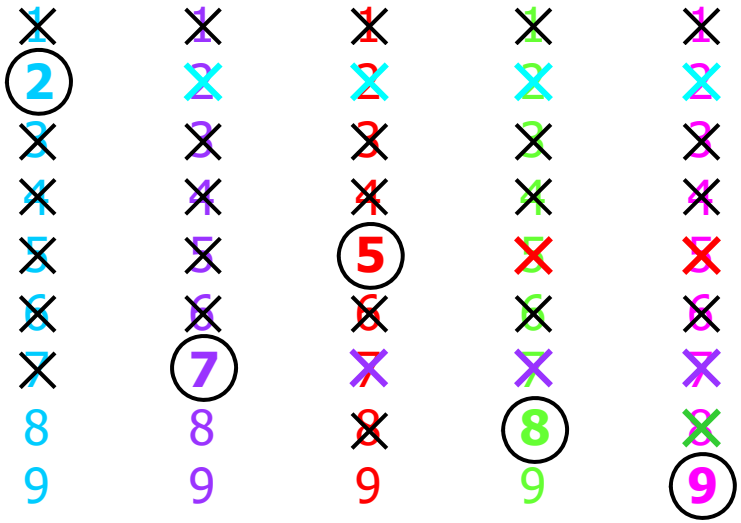
深度優先搜尋

- 基本的解法延伸前面產生各種排列的演算法，但是可以排進去的數字以及限制都改變了

		6	1		3	4		
		3			8			
5	4		7			1	2	
			2					4
7	5		3		9 4		7	
	3	7			5		4	2
			8			7		
		1	4		7	8		

2	7	6	1	5	3	4	8	9
		3			8			
5	4							
	5							
7								
	3							

…



深度優先搜尋

- 基本的解法延伸前面產生各種排列的演算法，但是可以排進去的數字以及限制都改變了

		6	1		3	4		
		3			8			
5	4		7			1	2	
			2					4
7	5		3		9 4		7	
	3	7			5		4	2
			8			7		
		1	4		7	8		

2	7	6	1	5	3	4	8	9
1		3			8			
5	4							

	5
7	
	3

...

~~1~~	~~1~~	~~1~~	~~1~~	~~1~~	1
2	~~2~~	~~2~~	~~2~~	~~2~~	~~2~~
~~3~~	~~3~~	~~3~~	~~3~~	~~3~~	~~3~~
~~4~~	~~4~~	~~4~~	~~4~~	~~4~~	~~4~~
~~5~~	~~5~~	5	~~5~~	~~5~~	~~5~~
~~6~~	~~6~~	~~6~~	~~6~~	~~6~~	~~6~~
~~7~~	7	~~7~~	~~7~~	~~7~~	~~7~~
8	8	~~8~~	8	~~8~~	~~8~~
9	9	9	9	9	9

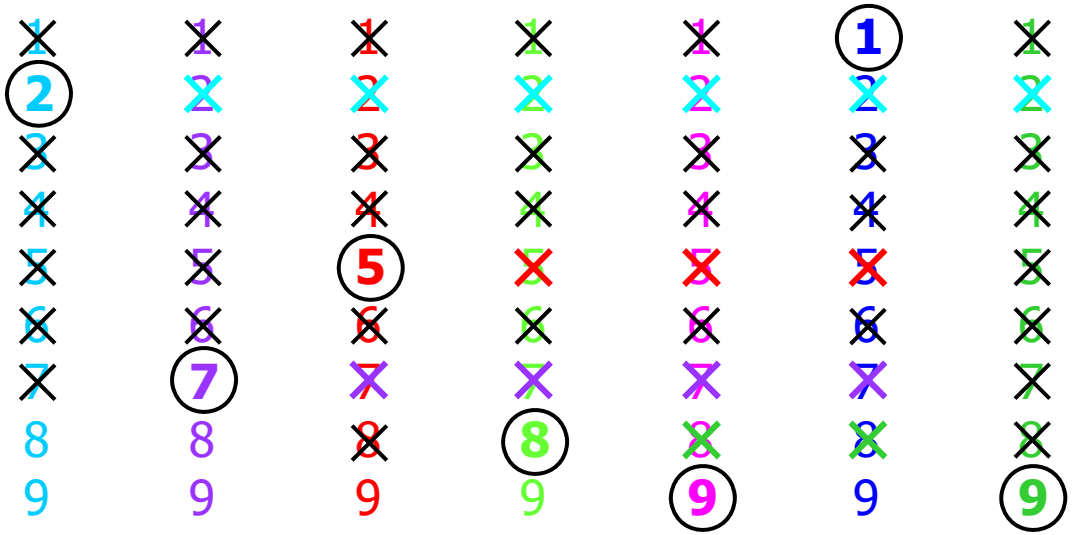
深度優先搜尋

- 基本的解法延伸前面產生各種排列的演算法，但是可以排進去的數字以及限制都改變了

		6	1		3	4		
		3			8			
5	4		7			1	2	
			2					4
7	5		3		9 4		7	
	3	7			5		4	2
			8			7		
		1	4		7	8		

2	7	6	1	5	3	4	8	9
1	9	3			8			
5	4							
	5							
7								
	3							

...



每一格可以填入資料的限制

- 填進每一格的新數值 value 都需要滿足下列三個限制
  - 每一行不會出現相同的數字

```
for (i=0; i<9; i++)
    if (value == board[i][icol]) return 0;
```

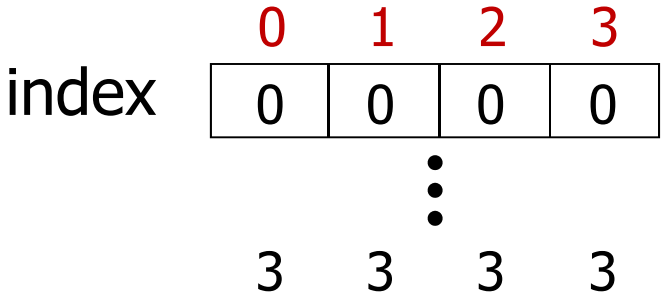
  - 每一列不會出現相同的數字

```
for (i=0; i<9; i++)
    if (value == board[irow][i]) return 0;
```

  - 每一個 3x3 的九宮格內不會出現相同的數字

```
for (i=irow/3*3; i<irow/3*3+3; i++)
    for (j=icol/3*3; j<icol/3*3+3; j++)
        if (value == board[i][j]) return 0;
```

以迴圈實作

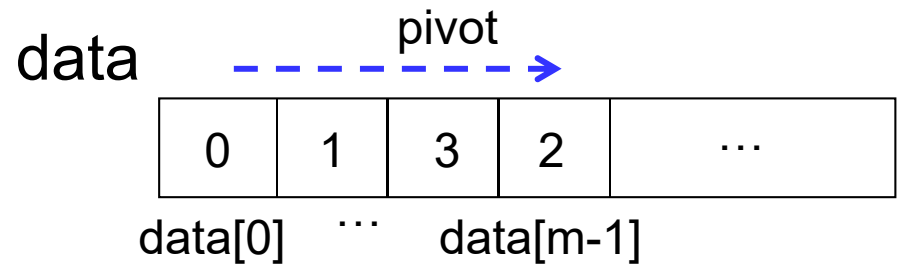


- **1**. 這一段程式固定由第 n-1 位數字開始加 1
  2. 可以加 1 的時候就結束了
  3. 到達 n-1 以後回復為 0， 迴圈執行到前一位數加 1
- 數獨要求 **1**. 由第 0 位開始表列, **2**. 可以加 1 且符合限制時要往後一位數開始表列, **3**. 到達 n-1 時回復為 0 以符合限制, 往前一位數加 1
- **for** 迴圈僅單向改變表列的位數，數獨雙向改變的要求需要一般化的迴圈
- 加 1 以後需要檢查是否符合限制，符合的話移到後一位數; 不符合繼續加 1

```
i=0;
while (1) {
    ip = i;
    while (index[i]<9) {
        index[i]++;
        if (valid(index, i)) { i++; break; }
    }
    if (ip==i) index[i--]=0;
}
```

```
void nextIndex(int index[], int n) {
    for (int i=n-1; i>0; i--)
        if (index[i]<n-1)
            { index[i]++; return; }
        else
            index[i] = 0;
    index[0]++;
}
```

以遞迴實作




```
void countUp(int n, int data[], int m, int pivot) {
    if (pivot>=m)
        print(data, m);
    else
        for (data[pivot]=0; data[pivot]<n; data[pivot]++)
            countUp(n, data, m, pivot+1);
}
```

```
void sudoku(int data[], int m, int pivot) {
    if (pivot>=m)
        print(data, m);
    else
        for (data[pivot]=1; data[pivot]<=9; data[pivot]++)
            if (valid(data, pivot))
                sudoku(data, m, pivot+1);
        data[pivot]=0;
}
```

```
sudoku(data, m, 0);
```

計算程式執行時間

- 這一小段程式運用 time.h 函式庫裡面的 clock() 這個函式來度量一個執行 60000000 次的空迴圈需要多少時間，呼叫 clock() 的時候會回傳一個型態是 clock_t 的整數，代表這個程式執行到目前為止所使用的 CPU 時間，單位是每秒 CLOCKS_PER_SEC，這個數值和你的機器目前是否同時執行很多其他程式是沒有太大關係的，所以執行前和執行後兩個數值相減就和執行時間是成正比的，如果執行的時間比較久，必須小心不要使 clock_t 型態的整數發生溢位

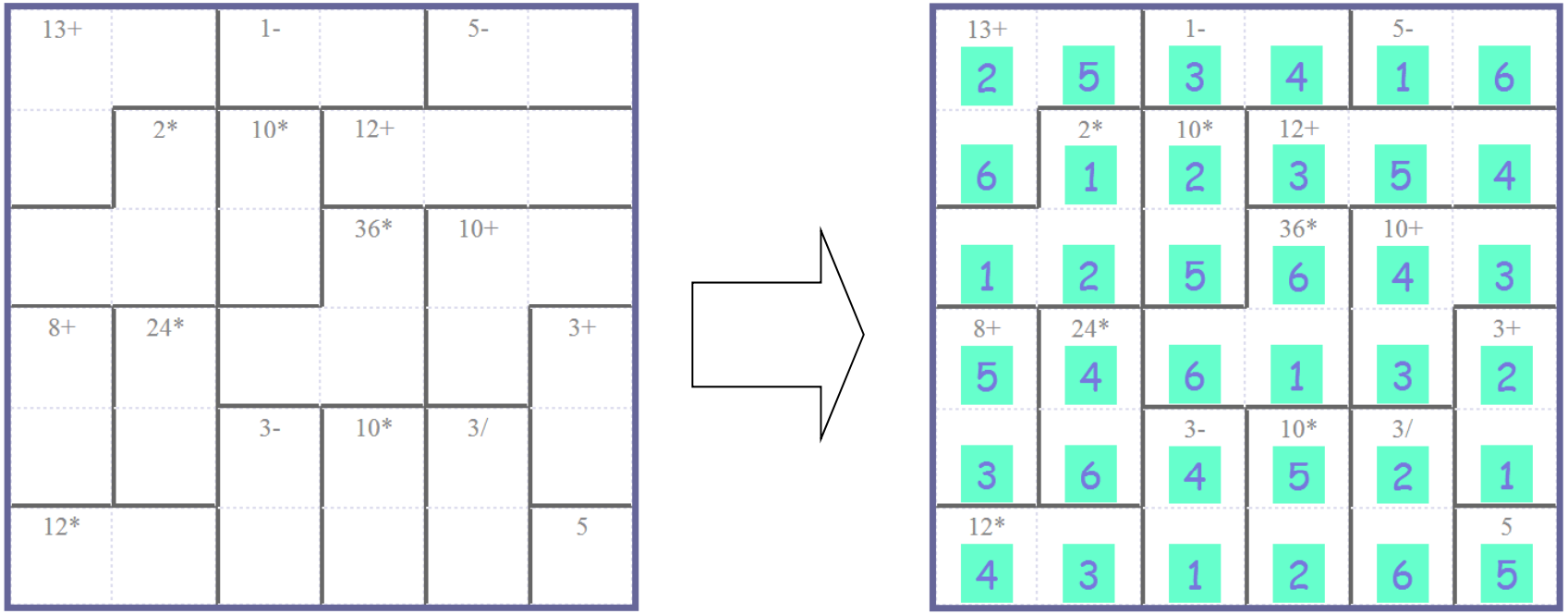
```
#include <stdio.h> /* printf */
#include <time.h>  /* clock_t, clock(), CLOCKS_PER_SEC, CLK_TCK */
int main(void) {
   int     i = 60000000;
   clock_t start, finish;
   double  duration;
   printf( "Time to do %ld empty loops is ", i );
   start = clock();
   while( i-- ) ;
   finish = clock();                          /* 或是 CLK_TCK */
   duration = (double)(finish - start) / CLOCKS_PER_SEC;
   printf( "%2.1f seconds\n", duration );
   return 0;
}
```

```
Time to do 60000000 empty loops is  0.2810 seconds
```

- Devcpp 5.x 會自動印出程式的執行時間 (秒)

延伸問題: Mathdoku

1. 4x4, 6x6, 8x8, …, nxn
2. The values in each cell are from the set {1, 2, …, n}
3. No repetition is allowed in each row and each column
4. In addition, the values in each non-regular subblocks should satisfy the labeled arithmetic constraints, e.g.
   13+ is satisfied by 2+5+6=13, 1- is satisfied by 4-3=1,
   36* is satisfied by 6*6*1=36, and 3/ is satisfied by 6/2=3



延伸問題 (cont'd)

下面這些網站上可以看到許多不同的 Sudoku 變化

1. Sudoku Dragon http://www.sudokudragon.com/sudokuvariants.htm
2. World Puzzle Federation
   http://www.worldpuzzle.org/championships/types-of-puzzles/wsc/
3. Sudoku Splash Zone
   http://www.sudokusplashzone.com/sudoku-variations/sudoku-variation-select.php
4. Hubpages
   http://somethingrandom.hubpages.com/hub/X-Different-Types-of-Sudoku-That-You-Should-Try