有限集合上遞移關係數量的計數問題：從暴力法到邊際效能優化

林辰訓

University
臺北市立大學
數學系

指導教授: 姚為成, 教授

June 1, 2023

Contents

大綱

本研究探討關係矩陣之遞移性檢測演算法，並計算數量為 n 之集合上的遞移關係總數。

- **製作動機**：為什麼會選這個題目
- **基礎理論**：以 $n = 2$ 為例，列舉 2^{n^2} 種關係並檢驗遞移性。
- **演算法實作**：從基礎暴力法 ($O(n^3)$) 到優化演算法。
- **效能瓶頸分析**：探討組合爆炸問題（Combinatorial Explosion），解釋為何當 $n > 5$ 時運算量激增。
- **實驗結果**：比對優化前後之加速倍率，並對照 OEIS A006905 序列數值。

基礎定義：遞移律 (Transitivity)

數學定義

設 R 為集合 A 上的一個關係。若對於所有的 $a, b, c \in A$:

$$(a, b) \in R \quad \text{且} \quad (b, c) \in R \implies (a, c) \in R$$

則稱 R 具有**遞移性**。

- **直觀理解**：如果有一條路徑可以從 a 走到 b，再從 b 走到 c，那麼必須存在一條「直達」的邊從 a 直接連向 c。
- **矩陣觀點**：在關係矩陣中，這代表若 $M_{ab} = 1$ 且 $M_{bc} = 1$，則 M_{ac} 必須為 1。

製作動機

從理論到實作的探索

- **驗證理論定義**：課本上的遞移律定義較抽象，我想透過程式窮舉所有可能，觀察遞移關係在集合中的真實分佈。
- **感受規模成長**：當集合元素增加時，關係總數以 2^{n^2} 劇烈成長。我想親自實作，體驗這種「組合爆炸」對計算帶來的挑戰。
- **尋求運算效率**：在面對數以億計的矩陣時，單純的暴力法是否足以應付？我想嘗試不同的程式邏輯來提升檢測速度。

製作動機

從理論到實作的探索

- **驗證理論定義**：課本上的遞移律定義較抽象，我想透過程式窮舉所有可能，觀察遞移關係在集合中的真實分佈。
- **感受規模成長**：當集合元素增加時，關係總數以 2^{n^2} 劇烈成長。我想親自實作，體驗這種「組合爆炸」對計算帶來的挑戰。
- **尋求運算效率**：在面對數以億計的矩陣時，單純的暴力法是否足以應付？我想嘗試不同的程式邏輯來提升檢測速度。

製作動機

從理論到實作的探索

- **驗證理論定義**：課本上的遞移律定義較抽象，我想透過程式窮舉所有可能，觀察遞移關係在集合中的真實分佈。
- **感受規模成長**：當集合元素增加時，關係總數以 2^{n^2} 劇烈成長。我想親自實作，體驗這種「組合爆炸」對計算帶來的挑戰。
- **尋求運算效率**：在面對數以億計的矩陣時，單純的暴力法是否足以應付？我想嘗試不同的程式邏輯來提升檢測速度。

實作前思考：實例分析 ($n = 2$)

針對集合 $A = \{1, 2\}$，其關係總數為 $2^{n^2} = 16$ 種。

序對數量	代表性組合	遞移性檢查
0 組 (1 項)	$\emptyset$	滿足 (Trivial)
1 組 (4 項)	$\{(1,1)\}, \{(1,2)\}, \ldots$	滿足
2 組 (6 項)	$\{(1,1),(1,2)\}, \ldots$	$\{(1,2),(2,1)\} \rightarrow$ 缺 $(1,1)$
3 組 (4 項)	$\{(1,1),(1,2),(2,2)\}, \ldots$	$\{(1,1),(1,2),(2,1)\} \rightarrow$ 缺 $(2,2)$
4 組 (1 項)	$\{(1,1),(1,2),(2,1),(2,2)\}$	滿足

非遞移反例判定

- $\{(1,2),(2,1)\} \implies$ 缺少 $(1,1)$
- $\{(1,2),(2,1),(2,2)\} \implies$ 缺少 $(1,1)$
- $\{(1,1),(1,2),(2,1)\} \implies$ 缺少 $(2,2)$

統計結論：

總關係數：16

非遞移項：3

遞移關係總數：13

實作前思考：從集合到矩陣

轉化動機

集合枚舉隨著 n 增加會面臨災難。改用 **矩陣**表示關係 R，可將遞移性判斷轉化為矩陣運算。

- **矩陣表示**：$a_{ij} = 1 \iff (i,j) \in R$。
- **遞移判定**：若 $a_{ik} = 1$ 且 $a_{kj} = 1$，則必須 $a_{ij} = 1$。

滿足遞移性範例		不具遞移性範例
$\{(1,1),(1,2)\}$ $\begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}$	vs	$\{(1,2),(2,1),(2,2)\}$ $\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$

Contents

遇到的瓶頸：窮舉法的困境

為了確保不遺漏任何遞移關係，我嘗試「暴力枚舉」所有可能的關係矩陣。

枚舉邏輯

- 一個 $n \times n$ 矩陣共有 n^2 格。
- 每格有 0, 1 兩種可能，總組合為 2^{n^2}。
- 使用整數 i 的位元依序填入矩陣。

實例：$n = 2, i = 6$

$i = 6 \implies (0110)_2$

$$\begin{bmatrix} a_{00} = 0 & a_{01} = 1 \\ a_{10} = 1 & a_{11} = 0 \end{bmatrix}$$

判定：存在 $(0, 1)$ 與 $(1, 0)$，但 $a_{00} = 0$（缺少 $(0, 0)$），故為非遞移關係。

效能瓶頸：組合爆炸

隨著 n 增加，計算量呈指數成長：

- $n = 4$：65,536 次 → 瞬間完成。
- $n = 5$：3,355 萬次 → 約 1 秒。
- $n = 6$：**687 億次** → **計算量激增 2048 倍**。

結論：
單純的「產生矩陣 → 檢查」邏輯在 $n = 6$ 時會撞上硬體性能牆。

Contents

演算法 A：基礎暴力法 (Naive)

這是最直觀的定義實作，使用三層嵌套迴圈檢查所有三元組 (i, k, j)。

Algorithm: is_transitive_naive(Matrix M, n)

- **For** $i = 0$ **to** $n - 1$:
- **For** $k = 0$ **to** $n - 1$:
- **If** $M[i][k] == 1$: // 若存在 $i \to k$
- **For** $j = 0$ **to** $n - 1$:
- **If** $M[k][j] == 1$ **and** $M[i][j] == 0$:
- **Return False** // 違反遞移律

Return True

- **複雜度**：$O(n^3)$
- **缺點**：內層迴圈無法利用現代處理器的並行特性。

演算法 B：位元並行優化法 (Bit-vector)

利用電腦字長一次處理整列的特性，將 $O(n)$ 的內層迴圈壓平。

Algorithm: is_transitive_opt(Rows R, n)

- **For** $i = 0$ **to** $n-1$:
- **For** $k = 0$ **to** $n-1$:
- **If** (k-th bit of $R[i]$ is 1): // 若存在 $i \to k$
- // 檢查：*Row[k]* 是否為 *Row[i]* 的子集
- **If** ($R[k]$ **AND** $R[i]$) $\neq R[k]$:
- **Return False**

Return True

- **複雜度**：實質降至 $O(n^2)$ (當 $n \leq 64$)
- **關鍵運算**：將 n 次判斷簡化為單一位元運算指令。

優化原理

1. **關係矩陣的位元向量化**
   - **Row 0**: 0 1 0 (0 → 1) → `dec:` 2
   - **Row 1**: 0 0 1 (1 → 2) → `dec:` 4
   - **Row 2**: 0 0 0 (無連向) → `dec:` 0
2. **判定邏輯 ($i = 0, k = 1$)**
   - 既然 0 → 1，則 1 能到的地方，0 也必須能到。
   - **條件**：Row 1 必須是 Row 0 的「子集」。
   - **運算**：`(R[1] & R[0]) == R[1]`

運算過程實測

- `R[1]:` 0 0 1
- `R[0]:` 0 1 0
- `AND` 結果：0 0 0

結果：000 ≠ 001 (判定失敗)
$\Longrightarrow$ 此關係非遞移。

優化效益

- **暴力法**：需用迴圈逐一檢查每個 j，耗時 $O(n)$。
- **位元優化**：利用 CPU `AND` 指令一次處理整列。速度提升約 n 倍。

位元優化實測：以 $i = 51$ 為例

$i = 51$ 的二進位映射為 `000 110 011`，對應矩陣：

Row 0: `000`　Row 1: `110`　Row 2: `011`

根據程式邏輯，系統會依序檢查矩陣中值為 1 的位置：

驗證步驟 (追蹤矩陣中的 1)

1. **驗證 (1,0)**：1 可以走到 0，需滿足 `Row 0 & Row 1 == Row 0`。
   運算：`000 & 110 = 000` (成立 ✓)
2. **驗證 (1,1)**：1 可以走到 1，需滿足 `Row 1 & Row 1 == Row 1`。
   運算：`110 & 110 = 110` (成立 ✓)
3. **驗證 (2,1)**：2 可以走到 1，需滿足 `Row 1 & Row 2 == Row 1`。
   運算：`110 & 011 = 010`
   結果：`010` $\neq$ `Row 1 (110)` **(不成立 x)**

結論：由於驗證 (2,1) 失敗，代表存在路徑 $2 \to 1 \to 2$ 但缺少直接關係 $2 \to 2$。

Contents

遭遇困難

- **運算量爆炸性成長**：當 $n = 5$ 時，C 語言不到 1 秒就跑完 3,355 萬個組合；但 $n = 6$ 直接飆升到 **687 億**個。即便 C 語言效率很高，這種「千倍等級」的差距還是讓普通電腦跑得非常吃力，這是我第一次親身體會到演算法複雜度的威力。
- **找不到規律走捷徑**：我原本想嘗試用遞迴的方法，看能不能用 $n - 1$ 的結果推導出 n 的數量。但後來發現「遞移性」只要矩陣中任一個位子改變，整個性質就得重新判定，沒辦法「偷懶」少算，只能老老實實地全部窮舉。
- **卡在語法細節與 Debug**：在實作過程中，我花了很多時間在處理模組瑣碎的語法。原本以為邏輯對了就沒事，但常常因為括號位置、位元運算優先權，或是陣列索引值算錯，導致跑出來的答案跟正確數列完全對不起來。為了找出這些小 bug，耗費了大量的精神去反覆檢查。

數據對照：OEIS A006905 數列

我將程式跑出來的結果與網路上 OEIS 數列對照，確認前 5 項完全正確：

n	總矩陣數量 (2^{n^2})	遞移關係數量 (我的結果)
1	2	2
2	16	13
3	512	171
4	65,536	3,994
5	33,554,432	154,303
6	**687 億**	**等待運算中...**

註：$n = 6$ 的計算量是 $n = 5$ 的 2048 倍，這就是為什麼電腦會卡住的原因。

數據對照：OEIS A006905 數列

A006905 as a simple table

n	a(n)
0	1
1	2
2	13
3	171
4	3994
5	154303
6	9415189
7	878222530
8	122207703623
9	24890747921947
10	7307450299510288
11	3053521546333103057
12	1797003559223770324237
13	1476062693867019126073312
14	1679239558149570229156802997
15	2628225174143857306623695576671
16	5626175867513779058707006016592954
17	16388270713364863943791979866838296851
18	64662720846908542794678859718227127212465

Contents

學到的技巧

在寫程式的過程中，我學到了幾個關鍵技巧：

- **位元運算的魔力**：以前課本說的 `<<` (左移) 和 `&` (位元與) 真的很實用。我用一個整數來代表矩陣的一列，這樣判斷遞移性時，速度比寫三層迴圈快非常多。
- **及時判斷 (Early Exit)**：不用把整個矩陣檢查完。只要發現有一個地方不符合 $a_{ik} = 1, a_{kj} = 1 \implies a_{ij} = 1$，就立刻跳出，這樣可以節省很多時間。
- **狀態空間的映射 (State Mapping)**：學習如何將一個超大的整數 i 透過位元偏移（Offset）映射回 $n \times n$ 的關係矩陣。這讓我也理解到，當狀態數達到 2^{n^2} 時，如何高效地在「整數索引」與「矩陣結構」間轉換是處理組合問題的核心。

Contents

照片

compiler

```
遞移關係計算實驗報告 (n=1 to 5)
----------------------------------------
N = 1 (總組合數: 2)
  [Naive] 數量: 2, 時間: 0.0000010 s
  [Optim] 數量: 2, 時間: 0.0000010 s
  >> 加速倍率: 1.00x

N = 2 (總組合數: 16)
  [Naive] 數量: 13, 時間: 0.0000020 s
  [Optim] 數量: 13, 時間: 0.0000010 s
  >> 加速倍率: 2.00x

N = 3 (總組合數: 512)
  [Naive] 數量: 171, 時間: 0.0000660 s
  [Optim] 數量: 171, 時間: 0.0000350 s
  >> 加速倍率: 1.89x

N = 4 (總組合數: 65536)
  [Naive] 數量: 3994, 時間: 0.0087950 s
  [Optim] 數量: 3994, 時間: 0.0029950 s
  >> 加速倍率: 2.94x

N = 5 (總組合數: 33554432)
  [Naive] 數量: 154303, 時間: 4.0182700 s
  [Optim] 數量: 154303, 時間: 0.8888960 s
  >> 加速倍率: 4.52x

N = 6 (總組合數: 68719476736)
```

Contents

參考資料

- 所有資料 (latex 開發碼/c code/照片) [網址] https://github.com/david910330d1d1d1d110-cyber/561
- 前 17 項遞移關係數量、相關證明 [網址] https://www.researchgate.net/publication/352383148_On_the_number_of_transitive_relations_on_a_set
- 其他關係的補充 [網址] https://www.youtube.com/watch?v=GvNGf9Gki7o
- C 的時間模組. [網址] https://pydoing.blogspot.com/2010/07/c-stdtime.html