NP 難題演算法系列 · 第 04 篇

N 皇后問題

N-Queens — The Gateway to Backtracking

回溯法的「Hello World」：學會在死路上及早回頭，一輩子受用

Contents

1 問題是什麼

定義 1.1 (N-Queens)**.** 在 $n \times n$ 棋盤上放 n 個皇后，使任兩個皇后不互相攻擊（不同列、不同行、不同對角線）。問：有幾種放法？或給出一種放法。

白話比喻

把它想成排班：n 個時段（列）各安排一位員工到某櫃檯（行），但有些組合會「打架」（同行、同對角線）。回溯法的精神是：**一格一格排，發現打架立刻回頭換人，而不是排完整張表才檢查**。

1.1 難度定位

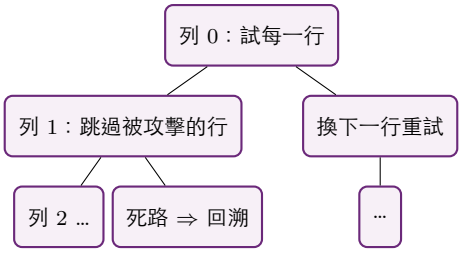
N-Queens 本身其實**不是** NP-Hard——任意 $n \geq 4$ 都保證有解，甚至有公式可以直接構造一組解。它的價值在於：

- 它是**回溯 + 剪枝**最乾淨的教學載體，而回溯正是攻打 3-SAT、著色、Hamiltonian 等真 NP-Hard 問題的基本武器。
- **計數**所有解（OEIS A000170）沒有已知多項式公式，$n = 27$ 的解數花了超級電腦數月才算出。
- 它的變形 N-Queens Completion（部分皇后已固定，能否補完？）已被證明 NP-Complete。

2 回溯法的骨架

2.1 核心觀察

每列恰好一個皇后 $\Rightarrow$ 解可表示為排列 $pos[0..n-1]$，$pos[r] =$ 第 r 列皇后所在行。逐列放置：



2.2 $O(1)$ 衝突檢查

天真寫法每次掃描已放的皇后要 $O(n)$。觀察三種攻擊各自的不變量：

攻擊方向	不變量	陣列大小
直行	c	n
主對角線 ↘	$r - c$（平移 $+n-1$）	$2n-1$
副對角線 ↙	$r + c$	$2n-1$

三個布林陣列，放皇后設真、回溯設假，檢查變 $O(1)$。

2.3 手算範例：$n = 4$

- 列 0 放行 0 $\rightarrow$ 列 1 只能放行 2 $\rightarrow$ 列 2 四行全被攻擊 $\Rightarrow$ **回溯**。
- 列 1 改放行 3 $\rightarrow$ 列 2 放行 1 $\rightarrow$ 列 3 全被攻擊 $\Rightarrow$ 回溯到底，列 0 換行。
- 列 0 放行 1 $\rightarrow$ 列 1 放行 3 $\rightarrow$ 列 2 放行 0 $\rightarrow$ 列 3 放行 2 $\Rightarrow$ **找到解** $(1, 3, 0, 2)$。

$n = 4$ 共 2 個解（互為鏡像）。本篇程式實測只拜訪 17 個節點，而天真枚舉 $4^4 = 256$ 種。

3 位元運算加速

把「哪些行被攻擊」壓進一個整數的位元：

- `cols`：被佔用的行。
- `diag1`：主對角線攻擊，傳給下一列時**整體左移一位**（對角線往右下延伸）。
- `diag2`：副對角線攻擊，傳給下一列時右移一位。
- 可放位置 `avail = full & ~(cols | diag1 | diag2)`，用 `avail & -avail` 逐一取出最低位的 1。

不用任何陣列、不用還原狀態（引數傳值），速度快約 8–10 倍（見執行結果 Example 4）。這招在所有「行/集合可用位元表示」的回溯題都通用。

4 完整 C++ 程式

包含：教學版回溯（三布林陣列 + 解的還原）、位元加速版、節點數統計、效能對比。編譯：

```
g++ -std=c++17 -O2 -Wall -Wextra -o nqueens nqueens.cpp
```

```cpp
// nqueens.cpp
// ============================================================================
// N-Queens：回溯法入門經典
//   1. 基本回溯：三個布林陣列 O(1) 衝突檢查
//   2. 位元運算加速版（competitive programming 風格）
//   3. 輸出一組具體解（棋盤圖）
//
// Compile: g++ -std=c++17 -O2 -Wall -Wextra -o nqueens nqueens.cpp
// ============================================================================
#include <iostream>
#include <vector>
#include <chrono>

using namespace std;

// ---------- 1. 基本回溯（教學版） ----------
class NQueensBasic {
    int n;
    vector<bool> col, d1, d2;     // 直行、主對角(r+c)、副對角(r-c+n-1)
    vector<int> pos;              // pos[row] = 該列皇后放的行
public:
    long long solutions = 0;
    long long nodes = 0;
    vector<int> firstSolution;

    void run(int n_) {
        n = n_;
        solutions = nodes = 0;
        col.assign(n, false);
        d1.assign(2 * n - 1, false);
        d2.assign(2 * n - 1, false);
        pos.assign(n, -1);
        firstSolution.clear();
        dfs(0);
    }
```

```
private:
    void dfs(int row) {
        ++nodes;
        if (row == n) {
            ++solutions;
            if (firstSolution.empty()) firstSolution = pos;
            return;
        }
        for (int c = 0; c < n; ++c) {
            if (col[c] || d1[row + c] || d2[row - c + n - 1]) continue;
                // 剪枝
            col[c] = d1[row + c] = d2[row - c + n - 1] = true;
            pos[row] = c;
            dfs(row + 1);
            col[c] = d1[row + c] = d2[row - c + n - 1] = false;
        }
    }
};

// ---------- 2. 位元運算版（每個位元 = 一行是否被攻擊） ----------
class NQueensBitmask {
    int n, full;
public:
    long long solutions = 0;
    long long count(int n_) {
        n = n_;
        full = (1 << n) - 1;
        solutions = 0;
        dfs(0, 0, 0);
        return solutions;
    }
private:
    void dfs(int cols, int diag1, int diag2) {
        if (cols == full) { ++solutions; return; }
        // 可放位置 = 不被三種攻擊覆蓋的位元
        int avail = full & ~(cols | diag1 | diag2);
        while (avail) {
            int bit = avail & (-avail);        // 取最低位的 1
            avail ^= bit;
            dfs(cols | bit, (diag1 | bit) << 1, (diag2 | bit) >> 1);
        }
    }
};

static void printBoard(const vector<int>& sol) {
    int n = (int)sol.size();
    for (int r = 0; r < n; ++r) {
        for (int c = 0; c < n; ++c)
            cout << (sol[r] == c ? " Q" : " .");
        cout << "\n";
    }
}

int main() {
    // ---- 範例 1：n=4 到 n=8 的解數 ----
    cout << "=== Example 1: solution counts ===\n";
    NQueensBasic nb;
    for (int n = 4; n <= 8; ++n) {
        nb.run(n);
```

```
94          cout << "n=" << n << " -> " << nb.solutions
95               << " solutions (search nodes=" << nb.nodes << ")\n";
96      }
97      cout << "\n";
98
99      // ---- 範例 2：印出 n=8 的第一組解 ----
100     cout << "=== Example 2: one solution for n=8 ===\n";
101     nb.run(8);
102     printBoard(nb.firstSolution);
103     cout << "\n";
104
105     // ---- 範例 3：剪枝的威力——比較節點數與天真枚舉 ----
106     cout << "=== Example 3: pruning power ===\n";
107     nb.run(8);
108     cout << "n=8: naive enumeration = 8^8 = 16,777,216 placements\n";
109     cout << "     backtracking visited only " << nb.nodes << " nodes\n\n";
110
111     // ---- 範例 4：位元版 vs 基本版效能 ----
112     cout << "=== Example 4: basic vs bitmask timing ===\n";
113     NQueensBitmask nq;
114     for (int n : {10, 12, 14}) {
115         auto t0 = chrono::steady_clock::now();
116         nb.run(n);
117         auto t1 = chrono::steady_clock::now();
118         long long c2 = nq.count(n);
119         auto t2 = chrono::steady_clock::now();
120         cout << "n=" << n << " solutions=" << nb.solutions
121              << " | basic "
122              << chrono::duration<double, milli>(t1 - t0).count() << " ms"
123              << " | bitmask "
124              << chrono::duration<double, milli>(t2 - t1).count() << " ms"
125              << " | agree=" << (nb.solutions == c2 ? "yes" : "NO!") <<
                    "\n";
126     }
127     return 0;
128 }
```

程式碼 1: nqueens.cpp — N-Queens 回溯與位元加速完整實作

5 執行結果與解讀

```
=== Example 1: solution counts ===
n=4 -> 2 solutions (search nodes=17)
n=5 -> 10 solutions (search nodes=54)
n=6 -> 4 solutions (search nodes=153)
n=7 -> 40 solutions (search nodes=552)
n=8 -> 92 solutions (search nodes=2057)

=== Example 2: one solution for n=8 ===
 Q . . . . . . .
 . . . . Q . . .
 . . . . . . . Q
 . . . . . Q . .
 . . Q . . . . .
 . . . . . . Q .
 . Q . . . . . .
 . . . Q . . . .

=== Example 3: pruning power ===
```

```
n=8: naive enumeration = 8^8 = 16,777,216 placements
     backtracking visited only 2057 nodes

=== Example 4: basic vs bitmask timing ===
n=10 solutions=724    | basic 2.28 ms   | bitmask 0.32 ms  | agree=yes
n=12 solutions=14200  | basic 71.8 ms   | bitmask 9.59 ms  | agree=yes
n=14 solutions=365596 | basic 2624 ms   | bitmask 304 ms   | agree=yes
```

筆記

兩個值得咀嚼的數字：(1) $n = 8$ 的搜尋空間 1600 萬，剪枝後只剩 2057 個節點——**回溯的本質是「及早發現不可行、成批地放棄」**。(2) $n = 6$ 的解數（4）比 $n = 5$（10）還少——解數並不單調，這種反直覺正是它沒有簡單公式的徵兆。

常見誤解

常見 bug：回溯後忘記**復原狀態**（把布林陣列設回假）。回溯法的黃金格式是「做選擇 → 遞迴 → **撤銷選擇**」三行一組，缺一不可。位元版之所以不會犯這錯，是因為狀態以傳值方式進遞迴，天然免疫。

6 回溯法的通用模板

```
void dfs(State s) {
    if (goal(s))    { record(s); return; }
    for (auto choice : choices(s)) {
        if (!feasible(s, choice)) continue;  // 剪枝
        apply(s, choice);                    // 做選擇
        dfs(s);                              // 遞迴
        undo(s, choice);                     // 撤銷（回溯）
    }
}
```

本系列的 3-SAT（DPLL）、Graph Coloring、Vertex Cover、Hamiltonian 全是這個模板加上各自的剪枝規則。**剪枝越準，樹越小**——演算法功力的差距全在 `feasible` 的設計。

7 實務應用

- **約束滿足問題（CSP）**：數獨、排課、拼圖求解器的骨架與 N-Queens 完全同構。
- **硬體測試**：N-Queens 常被用作平行運算與 FPGA 的基準測試（$n = 27$ 世界紀錄由 FPGA 集群完成）。
- **教學意義**：面試與競程的回溯題（全排列、組合、分割字串）都是本模板的直接應用。

8 練習題

1. 手畫 $n = 5$ 的搜尋樹前三層，數一數第一個解出現前回溯了幾次。
2. 利用鏡像對稱加速：第一列只枚舉前半行，解數乘 2（n 奇數時中間行特殊處理）。實測節點數減半。
3. 改成**找一組解就停**，$n = 30$ 能多快？（提示：位元版 + 提前 return。）
4. N-Rooks（只有直行攻擊）的解數是 $n!$——修改程式驗證 $n = 6$ 時是 720。

5. 挑戰：N-Queens Completion——先隨機固定 3 個合法皇后再補完。體會為什麼「部分固定」讓問題變 NP-Complete。

9 小結

版本	每節點成本	特點
天真枚舉 n^n	$O(n^2)$ 檢查	只能當反面教材
回溯 + 三陣列	$O(1)$ 檢查	教學首選，好懂好寫
位元回溯	$O(1)$、無陣列	快 8–10 倍，競賽首選

延伸閱讀：OEIS A000170（解數數列）；Knuth *The Art of Computer Programming* Vol. 4 的 Dancing Links（另一種精確覆蓋觀點）；Gent et al. (2017) 證明 N-Queens Completion 是 NP-Complete 的論文。