NP 難題演算法系列 · 第 08 篇

漢米爾頓路徑

Hamiltonian Path — Bitmask Dynamic Programming

每個點恰好經過一次：用位元把「走過哪些點」壓成一個整數，$n!$ 變 2^n

Contents

1 問題是什麼

定義 1.1 (Hamiltonian Path / Cycle)**. Hamiltonian Path**：圖中一條經過**每個頂點恰好一次**的路徑。首尾若還有邊相連即為 **Hamiltonian Cycle**。判定存在性皆為 NP-Complete（Karp 21；由 Vertex Cover 歸約）。

常見誤解

別跟 **Euler Path**（每條**邊**恰好一次）搞混！Euler 路徑有漂亮的度數判準（奇度點 0 或 2 個），$O(E)$ 可解；把「邊」換成「點」，難度立刻跳到 NP-Complete。一字之差，天堂地獄。

白話比喻

掃地機器人要走訪家裡每個房間**恰好一次**（不想重複吸同一間）；郵差要送 n 個包裹，每站去一次。有沒有這樣的順路走法？——這就是 Hamiltonian Path。若還要回到充電座，就是 Cycle。

2 核心技術：狀態壓縮 DP

2.1 暴力的絕望與 DP 的救贖

枚舉所有拜訪順序是 $O(n!)$：$n = 18$ 時 $18! \approx 6.4 \times 10^{15}$，宇宙熱寂前跑不完。關鍵觀察：

筆記

走到一半時，未來的可行性只取決於「**已經走過哪些點**」和「**現在站在哪**」——不在乎走過的**順序**。$(\{1, 3, 5\},\ 3)$ 這個狀態不管是 $1 \to 5 \to 3$ 還是 $5 \to 1 \to 3$ 走出來的，後續完全一樣。合併它們，$n!$ 條路徑坍縮成 $2^n \cdot n$ 個狀態。

2.2 狀態設計

用 n 位元整數 S 表示走過的點集（第 i 位 $=$ 點 i 走過沒）：

$$dp[S][v] = \text{「存在一條恰好走過} S \text{、結尾在} v \text{ 的路徑」（布林）}$$

初始：$dp[\{v\}][v] =$ 真（單點路徑）。　**轉移**：

$$dp[S \cup \{u\}][u] \ \mid= \ dp[S][v] \ \wedge \ (v, u) \in E \ \wedge \ u \notin S$$

答案：存在 v 使 $dp[\text{全集}][v]$ 為真。時間 $O(2^n n^2)$、空間 $O(2^n n)$。

2.3 三個實用細節

- **路徑還原**：記 $parent[S][v] =$ 轉移來源點，從終點沿 $S \setminus \{v\}$ 倒著走回去。
- **找 Cycle**：固定起點 0（環從哪開始都一樣），DP 只從 $dp[\{0\}][0]$ 起跑；最後檢查結尾 v 與 0 有邊。
- **位元技巧**：鄰接表也存成 bitmask，「v 的未走訪鄰居」= `adj[v] & ~S`，配合 `__builtin_ctz` 逐位取出。

常見誤解

本程式開發時真實踩過的 bug：找 Cycle 時若 DP 仍從**所有**單點起跑，$dp[\text{全集}][v]$ 為真時還原出來的路徑起點**不一定是 0**，直接接回 0 就錯了（立方體圖被誤判成沒有環）。教訓：**要求路徑從特定點出發，就只播那一顆種子**。

2.4 手算範例

菱形圖：邊 $0-1,\ 0-2,\ 1-2,\ 1-3,\ 2-3$。求 0 出發的 Hamiltonian Path：

S（二進位）	可行的 (S, v)	說明
0001	$(0001, 0)$	起點
0011	$(0011, 1)$	$0 \to 1$
0101	$(0101, 2)$	$0 \to 2$
0111	$(0111, 2), (0111, 1)$	$0 \to 1 \to 2$ 或 $0 \to 2 \to 1$
1011	$(1011, 3)$	$0 \to 1 \to 3$
1111	$(1111, 3), (1111, 1)$	如 $0 \to 1 \to 2 \to 3$、$0 \to 2 \to 3 \to 1$

$dp[1111][3]$ 為真 $\Rightarrow$ 存在，如 $0 \to 1 \to 2 \to 3$（用邊 $0-1, 1-2, 2-3$）。又 3 與 0 無邊，檢查其他結尾……$dp[1111][1]$：路徑 $0 \to 2 \to 3 \to 1$，而 $1-0$ 有邊 $\Rightarrow$ **Hamiltonian Cycle 也存在**：$0 \to 2 \to 3 \to 1 \to 0$。

3 完整 C++ 程式

包含：bitmask DP（Path 與 Cycle、含還原）、鏈 + 弦、立方體圖 Q_3（環 = 3 位元 Gray code）、Petersen 圖（著名的有 Path 無 Cycle）、$n = 18$ 效能展示。編譯：

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

```cpp
// hamiltonian_path.cpp
// ------------------------------------------------------------------
// Hamiltonian Path / Cycle：狀態壓縮 DP（bitmask DP）
//   dp[S][v] = 「是否存在一條路徑，恰好走過集合 S 的點，且結尾在 v」
//   轉移：dp[S | 1<<u][u] |= dp[S][v]  (v,u 相鄰, u 不在 S)
//   複雜度 O(2^n · n^2)，比 O(n!) 枚舉好非常多
//
// Compile: g++ -std=c++17 -O2 -Wall -Wextra -o hamiltonian_path
    hamiltonian_path.cpp
// ------------------------------------------------------------------
#include <iostream>
#include <vector>
#include <algorithm>
#include <chrono>

using namespace std;

class Hamiltonian {
    int n;
    vector<unsigned> adj;                    // bitmask 鄰接
    vector<vector<int>> parent;              // parent[S][v] =
        前一個點（還原用）
    vector<vector<char>> dp;
public:
    Hamiltonian(int n_) : n(n_), adj(n_, 0) {}
    void addEdge(int u, int v) {
        adj[u] |= 1u << v;
        adj[v] |= 1u << u;
    }

    // 找 Hamiltonian Path（起點任意）；回傳空 = 不存在
    vector<int> findPath() {
        buildDP(false);
        unsigned full = (1u << n) - 1;
        for (int v = 0; v < n; ++v)
            if (dp[full][v]) return reconstruct(full, v);
        return {};
```

```
    }

    // 找 Hamiltonian Cycle：固定起點 0（環的起點不影響存在性），
    // 尾點 v 需與 0 相鄰才能閉合
    vector<int> findCycle() {
        buildDP(true);                          // DP 只從點 0 起跑
        unsigned full = (1u << n) - 1;
        for (int v = 1; v < n; ++v)
            if (dp[full][v] && (adj[v] >> 0 & 1)) {
                auto path = reconstruct(full, v);   // 必以 0 開頭
                path.push_back(0);              // 閉合
                return path;
            }
        return {};
    }

    long long statesVisited = 0;
private:
    void buildDP(bool fixStartAtZero) {
        dp.assign(1u << n, vector<char>(n, 0));
        parent.assign(1u << n, vector<int>(n, -1));
        statesVisited = 0;
        if (fixStartAtZero)
            dp[1u][0] = 1;                      // 只允許從 0 出發
        else
            for (int v = 0; v < n; ++v)
                dp[1u << v][v] = 1;             // 單點路徑
        for (unsigned S = 1; S < (1u << n); ++S)
            for (int v = 0; v < n; ++v) {
                if (!dp[S][v]) continue;
                ++statesVisited;
                unsigned cand = adj[v] & ~S; // v 的鄰居中還沒走過的
                while (cand) {
                    int u = __builtin_ctz(cand);
                    cand &= cand - 1;
                    unsigned S2 = S | (1u << u);
                    if (!dp[S2][u]) {
                        dp[S2][u] = 1;
                        parent[S2][u] = v;
                    }
                }
            }
    }
    vector<int> reconstruct(unsigned S, int v) {
        vector<int> path;
        while (v != -1) {
            path.push_back(v);
            int p = parent[S][v];
            S &= ~(1u << v);
            v = p;
        }
        reverse(path.begin(), path.end());
        return path;
    }
};

static void printPath(const vector<int>& p) {
    if (p.empty()) { cout << "none\n"; return; }
    for (size_t i = 0; i < p.size(); ++i)
```

```
        cout << p[i] << (i + 1 < p.size() ? " -> " : "\n");
}

int main() {
    // ---- 範例 1：有 Path 沒 Cycle 的圖 ----
    cout << "=== Example 1: path exists, cycle doesn't ===\n";
    // 0-1-2-3-4 一條鏈 + 邊 1-3
    Hamiltonian h1(5);
    h1.addEdge(0,1); h1.addEdge(1,2); h1.addEdge(2,3); h1.addEdge(3,4);
    h1.addEdge(1,3);
    cout << "Hamiltonian path : ";
    printPath(h1.findPath());
    cout << "Hamiltonian cycle: ";
    printPath(h1.findCycle());
    cout << "\n";

    // ---- 範例 2：立方體圖 Q3（Path 和 Cycle 都有） ----
    cout << "=== Example 2: cube graph Q3 ===\n";
    Hamiltonian h2(8);
    for (int u = 0; u < 8; ++u)
        for (int b = 0; b < 3; ++b) {
            int v = u ^ (1 << b);            // 恰差一個位元 = 立方體的邊
            if (u < v) h2.addEdge(u, v);
        }
    cout << "Hamiltonian path : ";
    printPath(h2.findPath());
    cout << "Hamiltonian cycle: ";
    printPath(h2.findCycle());
    cout << "(cycle = 3-bit Gray code!)\n\n";

    // ---- 範例 3：Petersen 圖——著名的「有 Path 沒 Cycle」 ----
    cout << "=== Example 3: Petersen graph (famous non-Hamiltonian)
       ===\n";
    Hamiltonian h3(10);
    int petE[][2] = {{0,1},{1,2},{2,3},{3,4},{4,0},
                     {5,7},{7,9},{9,6},{6,8},{8,5},
                     {0,5},{1,6},{2,7},{3,8},{4,9}};
    for (auto& e : petE) h3.addEdge(e[0], e[1]);
    cout << "Hamiltonian path : ";
    printPath(h3.findPath());
    cout << "Hamiltonian cycle: ";
    printPath(h3.findCycle());
    cout << "\n";

    // ---- 範例 4：複雜度對比 O(2^n n^2) vs O(n!) ----
    cout << "=== Example 4: bitmask DP vs factorial enumeration ===\n";
    // n=18 的隨機圖
    Hamiltonian h4(18);
    unsigned seed = 987654321;
    auto nextRand = [&seed]() {
        seed = seed * 1103515245 + 12345;
        return (seed >> 16) & 0x7fff;
    };
    for (int u = 0; u < 18; ++u)
        for (int v = u + 1; v < 18; ++v)
            if (nextRand() % 100 < 30) h4.addEdge(u, v);
    auto t0 = chrono::steady_clock::now();
    auto path = h4.findPath();
    auto t1 = chrono::steady_clock::now();
```

```
    cout << "n=18: path " << (path.empty() ? "not found" : "FOUND")
         << " in " << chrono::duration<double, milli>(t1 - t0).count()
         << " ms (DP states with value=1: " << h4.statesVisited << ")\n";
    cout << "compare: 18! = 6,402,373,705,728,000 permutations\n";
    cout << "         2^18 * 18^2 = 84,934,656 DP transitions\n";
    if (!path.empty()) {
        cout << "path: ";
        printPath(path);
    }
    return 0;
}
```

程式碼 1: hamiltonian_path.cpp — 漢米爾頓路徑狀態壓縮 DP 完整實作

4 執行結果與解讀

```
=== Example 1: path exists, cycle doesn't ===
Hamiltonian path : 4 -> 3 -> 2 -> 1 -> 0
Hamiltonian cycle: none

=== Example 2: cube graph Q3 ===
Hamiltonian path : 7 -> 5 -> 4 -> 6 -> 2 -> 3 -> 1 -> 0
Hamiltonian cycle: 0 -> 4 -> 5 -> 7 -> 6 -> 2 -> 3 -> 1 -> 0
(cycle = 3-bit Gray code!)

=== Example 3: Petersen graph (famous non-Hamiltonian) ===
Hamiltonian path : 7 -> 5 -> 8 -> 6 -> 9 -> 4 -> 3 -> 2 -> 1 -> 0
Hamiltonian cycle: none

=== Example 4: bitmask DP vs factorial enumeration ===
n=18: path FOUND in 50.7 ms (DP states with value=1: 534587)
compare: 18! = 6,402,373,705,728,000 permutations
         2^18 * 18^2 = 84,934,656 DP transitions
path: 16 -> 3 -> 15 -> 2 -> 12 -> 10 -> 4 -> 7 -> 13 -> 6 -> 17 -> 8 -> 14 -> 11
    -> 9 -> 1 -> 5 -> 0
```

筆記

Example 2 的環 $0 \to 4 \to 5 \to 7 \to 6 \to 2 \to 3 \to 1 \to 0$ 寫成二進位是 $000, 100, 101, 111, 110, 010, 011, 001$——每步恰好翻一個位元，正是 **3 位元 Gray code**。「n 位 Gray code 存在」與「超立方體圖 Q_n 有 Hamiltonian Cycle」是同一句話。Example 3 的 Petersen 圖則是教科書級的反例：有 Path、無 Cycle（它是 hypohamiltonian 圖：自己不行，隨便刪一點就行）。

5 與 TSP 的關係

Hamiltonian Path 是 TSP 的「存在版」：TSP 問**最短**的走訪路線，這裡只問**有沒有**。把 dp 的布林值換成「最短距離」，轉移的 OR 換成 min，就得到下一篇的 Held–Karp 演算法——同一個狀態設計，兩道經典問題。

6 實務應用

- **基因組定序**：把 DNA 片段重疊圖走一遍恰好一次以重建序列（實務多轉成 Euler 路徑模型的 de Bruijn 圖，正因 Hamiltonian 太難）。

- **電路布線與鑽孔**：PCB 鑽孔機走訪每個孔位一次的路線規劃。
- **解謎**：騎士巡遊（Knight's Tour）= 棋盤跳躍圖的 Hamiltonian Path；許多一筆畫遊戲的本體。
- **Gray code 與硬體**：旋轉編碼器、K-map 化簡的排序都依賴 Q_n 的 Hamiltonian 結構。

7 練習題

1. 手算 K_4（完全圖）的 dp 表，數出 Hamiltonian Cycle 的數量（提示：$(4-1)!/2 = 3$）。
2. 把程式改成**計數**版本：有幾條不同的 Hamiltonian Path？（布林 OR 改加法。）
3. 加一個優化：若圖不連通或存在度 0 的點，直接回報不存在。實測隨機稀疏圖的加速。
4. 實作「指定起點 s 與終點 t」的 Hamiltonian Path 判定。
5. 挑戰：5×5 棋盤的騎士巡遊——建跳躍圖後用本程式找 Path（提示：$n = 25$ 需要 `uint32_t` 以上的 mask 與大量記憶體，想想怎麼分治或改用回溯 + Warnsdorff 啟發式）。

8 小結

方法	時間	空間	適用時機
枚舉排列	$O(n!\,n)$	$O(n)$	$n \le 12$
bitmask DP	$O(2^n n^2)$	$O(2^n n)$	$n \le 22$ 左右
回溯 + 啟發式	指數、無保證	$O(n)$	特殊結構圖（如騎士巡遊）
Euler 化建模	$O(E)$	$O(E)$	問題能改寫成走邊

延伸閱讀：Held & Karp (1962)、Bellman (1962) 同年獨立提出此 DP；Dirac ／ Ore 定理（度數夠大保證 Hamiltonian）；Björklund (2010) $O(1.657^n)$ 的代數突破。