NP 難題演算法系列 · 第 05 篇

圖著色問題

Graph Coloring — Backtracking & Greedy Heuristics

相鄰的點不能同色，最少要幾種顏色？從排課到暫存器分配都是它

Contents

1 問題是什麼

白話比喻

期末考排時段：兩門課若有共同學生就不能同時考。把課畫成點、衝突畫成邊，「安排考試時段」就變成「幫點塗顏色，相鄰不同色」。**最少需要的時段數 = 最少需要的顏色數 = 色數。**

定義 1.1 (k-著色與色數)**.** 無向圖 $G=(V,E)$ 的 **k-著色**是函數 $c: V \to \{1,\dots,k\}$ 使每條邊 (u,v) 滿足 $c(u) \neq c(v)$。使 G 可著色的最小 k 稱為**色數** $\chi(G)$。

1.1 難度定位

- $k=2$：等於判斷二部圖，BFS 一次 $O(V+E)$ 搞定。
- $k \geq 3$：**3-Coloring 是 NP-Complete**（由 3-SAT 歸約），又一個「2 到 3 的懸崖」。

- 近似也難：除非 P=NP，不存在能保證 $\chi(G)^{1-\varepsilon}$ 倍以內的多項式近似演算法。著色是**出了名地難近似**。
- 特例是綠洲：平面圖 $\chi \le 4$（四色定理）、區間圖可貪心求得精確色數、樹 $\chi \le 2$。

1.2 幾個基本事實

性質 1.1. $\chi(G) \ge \omega(G)$（最大團的大小）——團內兩兩相鄰，顏色不能重複。

性質 1.2. $\chi(G) \le \Delta(G)+1$（Δ = 最大度數）——貪心著色的保證；Brooks 定理進一步說除完全圖與奇環外 $\chi \le \Delta$。

性質 1.3. 奇環 $\chi = 3$、偶環 $\chi = 2$；完全圖 K_n 的 $\chi = n$。

2 演算法一：回溯 k-著色

逐點嘗試 $1..k$ 每種顏色，只要與已著色鄰居衝突就跳過；全部點著完即成功：

```
bool dfs(int u) {
    if (u == n) return true;                    // 全著完
    for (int c = 0; c < k; ++c) {
        if (!feasible(u, c)) continue;          // 剪枝：鄰居已用 c
        color[u] = c;
        if (dfs(u + 1)) return true;
        color[u] = -1;                          // 回溯
    }
    return false;
}
```

最壞 $O(k^n)$，但衝突剪枝讓實際樹小得多。**求色數**：k 從 1 開始遞增，第一個可行的 k 就是 $\chi(G)$（因為可行性對 k 單調）。

實務技巧

兩個常用加速（練習題會做）：(1) **點順序**——先著度數大的點（衝突早暴露）；(2) **對稱破除**——第一個點固定顏色 0、第二個點最多用到顏色 1……新顏色只在必要時開啟，砍掉顏色重命名造成的重複搜尋。

3 演算法二：貪婪著色（Welsh–Powell）

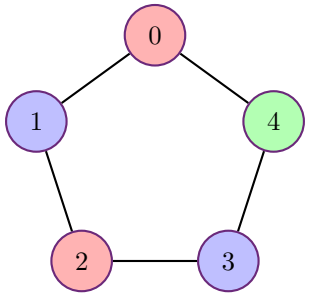
按**度數遞減**排序，逐點塗上「鄰居沒用過的最小顏色」：

$$T = O(V^2 + E) \qquad \text{保證用色} \le \Delta(G)+1$$

快但**不保證最優**；甚至存在讓貪心用到 $\Omega(n)$ 色而 $\chi = 2$ 的壞順序（皇冠圖）。實務上它是回溯法的好前鋒：先跑貪心拿到上界 k_0，回溯只需測 $k < k_0$。

3.1 手算範例：C5 五邊環

點 $0-1-2-3-4-0$。貪心（度數全為 2，按編號）：$0 \to$ 紅、$1 \to$ 藍、$2 \to$ 紅、$3 \to$ 藍、$4 \to$ ？——鄰居 3（藍）與 0（紅）都衝突，開第三色。$\chi(C_5) = 3$：奇環無法二著色（沿環交替塗色，環長為奇數時首尾必撞）。



4 完整 C++ 程式

包含：回溯 k-著色（含著色方案輸出）、遞增法求色數、Welsh–Powell 貪心、K3/C4/C5/Petersen 與排課實例。編譯：

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

```cpp
// graph_coloring.cpp
// ------------------------------------------------------------------
// Graph Coloring（圖著色）：
//   1. 回溯 + 剪枝：判斷 k 色可行並輸出著色
//   2. 色數（chromatic number）：從 k=1 遞增測試
//   3. 貪婪著色（Welsh-Powell：度數大者優先）——快速上界
//
// Compile: g++ -std=c++17 -O2 -Wall -Wextra -o graph_coloring
    graph_coloring.cpp
// ------------------------------------------------------------------
#include <iostream>
#include <vector>
#include <algorithm>
#include <numeric>

using namespace std;

// ---------- 1. 回溯 k-著色 ----------
class ColoringBacktrack {
    const vector<vector<int>>& adj;
    int n, k;
    vector<int> color;
public:
    long long nodes = 0;
    ColoringBacktrack(const vector<vector<int>>& g)
        : adj(g), n((int)g.size()) {}

    // 回傳 {可行?, 著色方案}
    pair<bool, vector<int>> solve(int k_) {
        k = k_;
        color.assign(n, -1);
        nodes = 0;
        bool ok = dfs(0);
        return {ok, color};
    }
private:
    bool feasible(int u, int c) {
        for (int v : adj[u])
            if (color[v] == c) return false;
        return true;
    }
```

```
    bool dfs(int u) {
        ++nodes;
        if (u == n) return true;
        for (int c = 0; c < k; ++c) {
            if (!feasible(u, c)) continue;          // 剪枝
            color[u] = c;
            if (dfs(u + 1)) return true;
            color[u] = -1;
        }
        return false;
    }
};

// ---------- 2. 色數：遞增 k ----------
int chromaticNumber(const vector<vector<int>>& g) {
    ColoringBacktrack cb(g);
    for (int k = 1; k <= (int)g.size(); ++k)
        if (cb.solve(k).first) return k;
    return (int)g.size();
}

// ---------- 3. 貪婪著色（Welsh-Powell 順序） ----------
pair<int, vector<int>> greedyColoring(const vector<vector<int>>& g) {
    int n = (int)g.size();
    vector<int> order(n);
    iota(order.begin(), order.end(), 0);
    sort(order.begin(), order.end(), [&](int a, int b) {
        return g[a].size() > g[b].size();           // 度數遞減
    });
    vector<int> color(n, -1);
    int used = 0;
    for (int u : order) {
        vector<bool> ban(n, false);
        for (int v : g[u])
            if (color[v] != -1) ban[color[v]] = true;
        int c = 0;
        while (ban[c]) ++c;
        color[u] = c;
        used = max(used, c + 1);
    }
    return {used, color};
}

static vector<vector<int>> buildGraph(int n, const vector<pair<int,int>>&
    edges) {
    vector<vector<int>> g(n);
    for (auto [u, v] : edges) {
        g[u].push_back(v);
        g[v].push_back(u);
    }
    return g;
}

static void report(const string& name, const vector<vector<int>>& g) {
    ColoringBacktrack cb(g);
    int chi = chromaticNumber(g);
    auto [greedyK, gc] = greedyColoring(g);
    auto [ok, col] = cb.solve(chi);
    cout << name << ": chromatic number = " << chi
```

```
          << " | greedy used = " << greedyK
          << (greedyK > chi ? "  (greedy overshoots!)" : "") << "\n";
    if (ok) {
        cout << "  optimal coloring:";
        for (int i = 0; i < (int)col.size(); ++i)
            cout << " v" << i << "=c" << col[i];
        cout << "\n";
    }
    (void)gc;
}

int main() {
    // ---- 範例 1：三角形 K3（色數 3） ----
    cout << "=== Example 1: triangle K3 ===\n";
    report("K3", buildGraph(3, {{0,1},{1,2},{2,0}}));
    cout << "\n";

    // ---- 範例 2：偶環 C4 vs 奇環 C5 ----
    cout << "=== Example 2: even cycle C4 (bipartite) vs odd cycle C5
        ===\n";
    report("C4", buildGraph(4, {{0,1},{1,2},{2,3},{3,0}}));
    report("C5", buildGraph(5, {{0,1},{1,2},{2,3},{3,4},{4,0}}));
    cout << "\n";

    // ---- 範例 3：Petersen 圖（著名例子，色數 3） ----
    cout << "=== Example 3: Petersen graph ===\n";
    vector<pair<int,int>> pet = {
        {0,1},{1,2},{2,3},{3,4},{4,0},          // 外圈
        {5,7},{7,9},{9,6},{6,8},{8,5},          // 內五角星
        {0,5},{1,6},{2,7},{3,8},{4,9}           // 輻條
    };
    report("Petersen", buildGraph(10, pet));
    cout << "\n";

    // ---- 範例 4：排課情境（衝突圖著色 = 時段分配） ----
    cout << "=== Example 4: exam scheduling as coloring ===\n";
    // 6 門課，邊 = 有共同學生（不能同時段）
    // 課程: 0=微積分 1=線代 2=物理 3=化學 4=程設 5=英文
    vector<pair<int,int>> conflicts = {
        {0,1},{0,2},{1,2},{1,3},{2,4},{3,4},{3,5},{4,5}
    };
    auto g = buildGraph(6, conflicts);
    int chi = chromaticNumber(g);
    ColoringBacktrack cb(g);
    auto [ok, col] = cb.solve(chi);
    const char* names[] =
       {"Calculus","LinAlg","Physics","Chem","Prog","English"};
    cout << "6 courses, 8 conflicts -> minimum time slots = " << chi <<
        "\n";
    if (ok)
        for (int i = 0; i < 6; ++i)
            cout << "  slot " << col[i] << ": " << names[i] << "\n";
    return 0;
}
```

程式碼 1: graph_coloring.cpp — 圖著色回溯與貪心完整實作

5 執行結果與解讀

```
=== Example 1: triangle K3 ===
K3: chromatic number = 3 | greedy used = 3
  optimal coloring: v0=c0 v1=c1 v2=c2

=== Example 2: even cycle C4 (bipartite) vs odd cycle C5 ===
C4: chromatic number = 2 | greedy used = 2
  optimal coloring: v0=c0 v1=c1 v2=c0 v3=c1
C5: chromatic number = 3 | greedy used = 3
  optimal coloring: v0=c0 v1=c1 v2=c0 v3=c1 v4=c2

=== Example 3: Petersen graph ===
Petersen: chromatic number = 3 | greedy used = 3
  optimal coloring: v0=c0 v1=c1 v2=c0 v3=c1 v4=c2 v5=c1 v6=c0 v7=c2 v8=c2 v9=c1

=== Example 4: exam scheduling as coloring ===
6 courses, 8 conflicts -> minimum time slots = 3
  slot 0: Calculus    slot 1: LinAlg     slot 2: Physics
  slot 0: Chem        slot 1: Prog       slot 2: English
```

筆記

Example 4 是著色的「應用原型」：6 門課、8 組衝突，色數 3 表示**三個時段就夠**，且程式直接給出可行的時段表。任何「資源互斥 $\Rightarrow$ 分組」的問題都能套這個流程：建衝突圖 $\rightarrow$ 求（近似）著色 $\rightarrow$ 每色一組。

6 實務應用

- **編譯器暫存器分配**：變數為點、生命週期重疊為邊，k 個暫存器 $= k$-著色；塗不進去的變數 spill 到記憶體。Chaitin (1981) 的經典演算法就是圖著色。
- **排課排考**：本篇 Example 4 的直接放大版，大學排考系統的核心。
- **無線電頻譜分配**：地理相鄰的基地台不能用同頻段——頻段 $=$ 顏色。
- **數獨**：81 個格子的圖著色（9 色），行、列、宮相鄰。
- **製程排程**：互斥資源的任務分批，每批一色。

7 練習題

1. 手算輪圖 W_5（C5 加中心點連向全部）的色數。
2. 實作「對稱破除」：第 u 個點只准用顏色 $0..\min(u, k-1)$。實測 Petersen 圖的節點數變化。
3. 構造一個圖與一種點順序，讓貪心用 4 色而 $\chi = 2$（提示：皇冠圖 crown graph）。
4. 把回溯的點順序改成「度數遞減」，比較 Petersen 圖上的節點數。
5. 挑戰：實作 DSATUR（動態飽和度優先）貪心，跟 Welsh–Powell 在隨機圖上比誰用色少。

8 小結

方法	時間	品質	適用時機
BFS 二著色	$O(V+E)$	精確（$k=2$）	判二部圖
回溯	最壞 $O(k^n)$	精確	$n \lesssim 30$ 或圖稀疏
Welsh–Powell 貪心	$O(V^2+E)$	$\le \Delta+1$ 色	要快、當上界
DSATUR	$O(V^2)$	通常更少色	實務首選貪心

延伸閱讀：四色定理（Appel & Haken 1976，第一個電腦輔助證明）；Brooks 定理；Chaitin (1981) Register Allocation via Coloring；Lewis *A Guide to Graph Colouring*。