IMO2007P3

Biên soạn bởi quỷ thuật Gemini
do thằng BirdBB chưa giải được

Ngày 13 tháng 6 năm 2025

Mục lục

1 Phân tích Bài toán và Thuật giải **2**
1.1 Phát biểu bài toán 2
1.2 Thuật giải 2

2 Giải thích Mã nguồn C++ **3**
2.1 Cấu trúc Dữ liệu và Biến Toàn cục 3
2.2 Hàm `find_max_clique_recursive` 3
2.3 Hàm `get_max_clique` 3
2.4 Hàm `main` 4

3 Test Case Mẫu Bổ Sung **5**
3.1 Input 5
3.2 Output Dự kiến 5
3.3 Kiểm tra Kết quả 5

1 Phân tích Bài toán và Thuật giải

1.1 Phát biểu bài toán

Bài toán có thể được mô hình hóa bằng lý thuyết đồ thị. Các thí sinh là các đỉnh (V), và quan hệ bạn bè là các cạnh (E). Một "nhóm bạn bè"(clique) là một tập hợp các đỉnh mà trong đó hai đỉnh bất kỳ đều có cạnh nối với nhau. Cỡ của clique là số đỉnh trong đó.

Giả thiết: Cỡ của clique lớn nhất trong đồ thị, ký hiệu là $\omega(G)$, là một số chẵn, $\omega(G) = 2k$.

Yêu cầu: Chứng minh rằng có thể phân hoạch tập đỉnh V thành hai tập R_1 và R_2 ($R_1 \cup R_2 = V$, $R_1 \cap R_2 = \emptyset$) sao cho cỡ của clique lớn nhất trong hai phòng bằng nhau. Tức là, $\omega(G[R_1]) = \omega(G[R_2])$.

1.2 Thuật giải

Thuật giải được sử dụng trong mã nguồn là một hiện thực hóa của phương pháp chứng minh mang tính xây dựng cho bài toán này. Ý tưởng chính là bắt đầu từ một trạng thái mất cân bằng và di chuyển dần dần các đỉnh để đạt tới trạng thái cân bằng.

1. **Bước 1: Tìm một Clique Lớn Nhất**
   Đầu tiên, ta phải tìm một clique bất kỳ có cỡ lớn nhất trong đồ thị, gọi clique này là C. Theo giả thiết, $|C| = 2k$.

2. **Bước 2: Phân hoạch Ban đầu**
   Ta khởi tạo hai phòng (hai tập đỉnh) như sau:

   - Phòng 1 (R_1) chứa tất cả các đỉnh thuộc clique C.
   - Phòng 2 (R_2) chứa tất cả các đỉnh còn lại của đồ thị ($V \setminus C$).

   Ở trạng thái này, cỡ clique lớn nhất trong phòng 1 là $2k$, trong khi phòng 2 có cỡ clique nhỏ hơn hoặc bằng $2k$. Thường thì hai giá trị này sẽ không bằng nhau.

3. **Bước 3: Quá trình Di chuyển Lặp**
   Ta thực hiện một vòng lặp, trong mỗi bước lặp, ta lấy một đỉnh từ clique C (mà hiện đang ở trong phòng 1) và chuyển nó sang phòng 2. Quá trình này được lặp lại cho đến khi tất cả các đỉnh của C đều được chuyển sang phòng 2.

4. **Bước 4: Phân tích sự Thay đổi và Điều kiện Dừng**
   Đây là mấu chốt của chứng minh. Gọi $k_{1,i}$ và $k_{2,i}$ là cỡ clique lớn nhất trong phòng 1 và phòng 2 sau khi đã di chuyển i đỉnh.

   - Chuỗi $k_{1,i}$ sẽ giảm một cách tuần tự $(2k, 2k-1, \ldots, 0)$.
   - Chuỗi $k_{2,i}$ sẽ không giảm, và ở mỗi bước nó chỉ có thể giữ nguyên hoặc tăng lên 1.

   Do đó, hiệu số $D_i = k_{1,i} - k_{2,i}$ sẽ giảm dần từ một số không âm xuống một số âm. Vì D_i là một chuỗi số nguyên và mỗi bước chỉ thay đổi một lượng nhỏ, nó chắc chắn sẽ đi qua giá trị 0.

   Khi $D_i = 0$, ta có $k_{1,i} = k_{2,i}$. Đây chính là trạng thái cân bằng mà ta cần tìm. Ta dừng thuật toán và đưa ra kết quả phân hoạch tại bước đó.

2 Giải thích Mã nguồn C++

2.1 Cấu trúc Dữ liệu và Biến Toàn cục

```
int n;
bool adj[65][65];
vector<int> best_clique_so_far;
```

- `int n;`: Lưu trữ số lượng đỉnh (thí sinh) của đồ thị.
- `bool adj[65][65];`: Ma trận kề để biểu diễn đồ thị. `adj[i][j] = true` nếu có cạnh nối giữa đỉnh i và j. Ma trận kề cho phép kiểm tra quan hệ bạn bè trong thời gian $O(1)$, rất hiệu quả cho thuật toán tìm clique.
- `vector<int> best_clique_so_far;`: Biến toàn cục dùng để lưu trữ clique lớn nhất tìm được trong quá trình chạy đệ quy. Việc dùng biến toàn cục giúp tránh phải truyền qua lại vector này giữa các lời gọi hàm đệ quy.

2.2 Hàm `find_max_clique_recursive`

Đây là hàm cốt lõi, hiện thực thuật toán quay lui (backtracking) để tìm clique lớn nhất.

```
void find_max_clique_recursive(const vector<int>& potential_nodes, vector<
    int> current_clique)
```

- **Tham số**:
  - `potential_nodes`: Danh sách các đỉnh "ứng viên"có thể được thêm vào clique hiện tại để mở rộng nó.
  - `current_clique`: Clique đang được xây dựng trong nhánh đệ quy hiện tại.
- **Logic hoạt động**:
  - **Cập nhật kết quả**: Dòng `if (current_clique.size() > best_clique_so_far.size())` kiểm tra nếu clique vừa xây dựng xong lớn hơn kết quả tốt nhất đã lưu, thì cập nhật lại.
  - **Vòng lặp và Đệ quy**: Duyệt qua các đỉnh ứng viên `v`. Với mỗi `v`, hàm tạo ra một tập ứng viên mới (`new_potential_nodes`) gồm các đỉnh kề với `v` trong số các ứng viên còn lại. Sau đó, nó gọi đệ quy với `v` được thêm vào clique.
  - **Cắt tỉa (Pruning)**: Dòng `if (current_clique.size() + (potential_nodes.size() - i) <= best_clique_so_far.size())` là một kỹ thuật tối ưu hóa cực kỳ quan trọng. Nó giúp dừng sớm việc tìm kiếm trên một nhánh nếu nhận thấy nhánh đó không thể cho ra kết quả tốt hơn kết quả đã có.

2.3 Hàm `get_max_clique`

Đây là hàm bao (wrapper) giúp việc gọi hàm đệ quy trở nên gọn gàng hơn.

```
vector<int> get_max_clique(const vector<int>& nodes)
```

Hàm này nhận vào một tập các đỉnh, khởi tạo/xóa trắng biến toàn cục `best_clique_so_far`, và bắt đầu quá trình tìm kiếm đệ quy.

2.4 Hàm `main`

Hàm chính điều khiển toàn bộ luồng của chương trình.

1. **Khởi tạo và Đọc Input**:

```
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    freopen("b.inp","r",stdin);
    freopen("b.out","w",stdout);

    cin >> n;
    cin.ignore();

```

   Các dòng đầu tiên dùng để tối ưu hóa tốc độ nhập xuất. Các dòng `freopen` dùng để đọc từ file `b.inp` và ghi ra file `b.out`. Phần còn lại đọc dữ liệu đầu vào và xây dựng ma trận kề `adj`.

2. **Tìm Clique Lớn Nhất Toàn Cục** C:

```
    vector<int> all_nodes(n);
    iota(all_nodes.begin(), all_nodes.end(), 1);

    vector<int> C = get_max_clique(all_nodes);
    cout << C.size() << endl;

```

   Đoạn này tạo ra một vector chứa tất cả các đỉnh từ 1 đến n, sau đó gọi `get_max_clique` để tìm clique lớn nhất C và in ra kích thước của nó.

3. **Thực hiện Thuật toán Phân phòng**:

```
    vector<int> room1 = C;
    vector<int> room2;

    for (size_t i = 0; i <= C.size(); ++i) {
        int size1 = get_max_clique(room1).size();
        int size2 = get_max_clique(room2).size();

        if (size1 == size2) {
        }

        if (i < C.size()) {
        }
    }

```

   Đây là phần hiện thực hóa thuật toán đã mô tả ở trên. Nó khởi tạo `room1` và `room2`, sau đó lặp qua các thành viên của C, di chuyển từng người một. Trong mỗi lần lặp, nó tính lại cỡ clique lớn nhất của hai phòng và kiểm tra xem chúng đã bằng nhau chưa. Nếu bằng nhau, nó in kết quả và kết thúc.

3 Test Case Mẫu Bổ Sung

Ta xét một đồ thị có 6 đỉnh, trong đó clique lớn nhất có cỡ 4 (là một số chẵn).

3.1 Input

```
6
2 3 4
1 3 4
1 2 4 5
1 2 3 6
3
4
```

3.2 Output Dự kiến

Dựa trên thuật toán, một trong những kết quả hợp lệ có thể là:

```
4
1 2
3 4 5 6
2
```

3.3 Kiểm tra Kết quả

- **Dòng 1**: Cỡ clique lớn nhất của toàn bộ đồ thị là 4 (chính là clique $\{1, 2, 3, 4\}$). Điều này đúng.
- **Dòng 2**: Phòng 1 chứa các đỉnh $\{1, 2\}$. Clique lớn nhất trong phòng này là chính nó, có cỡ 2.
- **Dòng 3**: Phòng 2 chứa các đỉnh $\{3, 4, 5, 6\}$. Ta cần tìm clique lớn nhất trong tập này.
  - Đỉnh 3 là bạn của 4, 5.
  - Đỉnh 4 là bạn của 3, 6.
  - Đỉnh 5 là bạn của 3.
  - Đỉnh 6 là bạn của 4.
  - Các clique có thể có là: $\{3, 5\}$, $\{4, 6\}$. Cả hai đều có cỡ 2.

  Do đó, cỡ clique lớn nhất trong phòng 2 là 2.
- **Dòng 4**: Cỡ lớn nhất chung của hai phòng là 2.

Kết quả này hoàn toàn hợp lệ vì $2 = 2$. Trạng thái này đạt được trong quá trình thuật toán di chuyển các đỉnh $\{3, 4\}$ từ clique ban đầu sang phòng 2.