Graph Theory in Competitive Programming
Lý Thuyết Đồ Thị Trong Lập Trình Thi Đấu

Nguyễn Quản Bá Hồng*

Ngày 3 tháng 8 năm 2025

Tóm tắt nội dung

This text is a part of the series *Some Topics in Advanced STEM & Beyond*:
URL: https://nqbh.github.io/advanced_STEM/.
Latest version:

- *Graph Theory in Competitive Programming – Lý Thuyết Đồ Thị Trong Lập Trình Thi Đấu.*
  PDF: URL: .pdf.
  TEX: URL: .tex.
- *Olympiad in Informatics & Association for Computing Machinery–International Collegiate Programming Contest – Olympic Tin Học Sinh Viên OLP & ICPC.*
  PDF: URL: https://github.com/NQBH/advanced_STEM_beyond/blob/main/OLP_ICPC/NQBH_OLP_ICPC.pdf.
  TEX: URL: https://github.com/NQBH/advanced_STEM_beyond/blob/main/OLP_ICPC/NQBH_OLP_ICPC.tex.
- Codes:
  - C: https://github.com/NQBH/advanced_STEM_beyond/tree/main/OLP_ICPC/C.
  - C++: https://github.com/NQBH/advanced_STEM_beyond/tree/main/OLP_ICPC/C++.
  - Java: https://github.com/NQBH/advanced_STEM_beyond/tree/main/OLP_ICPC/Java.
  - Python: https://github.com/NQBH/advanced_STEM_beyond/tree/main/OLP_ICPC/Python.

Mục lục

1 Breadth-First Search (BFS) – Tìm Kiếm Ưu Tiên Theo Chiều Rộng

Resources – Tài nguyên.

1. Nguyễn Châu Khanh. VNOI/BFS (breadth-first search).

2 Depth-First Search (DFS) – Tìm Kiếm Ưu Tiên Theo Chiều Sâu

Resources – Tài nguyên.

*A scientist- & creative artist wannabe, a mathematics & computer science lecturer of Department of Artificial Intelligence & Data Science (AIDS), School of Technology (SOT), UMT Trường Đại học Quản lý & Công nghệ TP.HCM, Hồ Chí Minh City, Việt Nam.
E-mail: nguyenquanbahong@gmail.com & hong.nguyenquanba@umt.edu.vn. Website: https://nqbh.github.io/. GitHub: https://github.com/NQBH.

3 Biconnected Component – Thành Phần Song Liên Thông

Resources – Tài nguyên.

1. Vietcodes's blog/thành phần song liên thông.

2. Wikipedia/biconnected component.

Định nghĩa 1 (Đỉnh khớp)**.** *Cho $G = (V, E)$ là 1 đồ thị vô hướng.* Đỉnh khớp (cut vertex) *là 1 đỉnh $v \in V$ mà nếu xóa đi sẽ làm tăng số thành phần liên thông của G.*

Intuition. Có vài cách mường tượng để hiểu nom na khái niệm đỉnh khớp:

1. Đỉnh khớp của 1 đồ thị vô hướng giống như 1 môi giới trung gian để đảm bảo sự liên lạc giữa 2 người không thể giao tiếp trực tiếp với nhau, i.e., nếu mất người môi giới trung gian thì 2 người đó sẽ không thể liên lạc với nhau, cả trực tiếp lẫn gián tiếp, được nữa.

2. Tưởng tượng có 2 hòn đảo ngăn cách nhau bởi 1 con sông lớn & có 1 tảng đá hoặc 1 mô đất đủ nhỏ giữa sông tượng trưng cho đỉnh khớp. Chiều rộng lòng sông đủ lớn để 1 người không thể nhảy trực tiếp từ bên đảo này sang đảo kia, nhưng nếu từ 1 trong 2 hồn đảo, người đó nhảy gián tiếp qua tảng đá hoặc 1 mô đất đủ nhỏ kia thì sẽ qua được hòn đảo còn lại, nên đảm bảo được tính liên thông, i.e., khả năng di chuyển qua lại giữa 2 hòn đảo. Nếu chẳng may tảng đá hoặc 1 mô đất đó mất đi, i.e., bỏ đi đỉnh khớp, thì người đó không thể di chuyển qua lại giữa 2 hòn đảo nữa, khi đó 2 hòn đảo trở thành 2 thành phần liên thông tách biệt nhau.

Định nghĩa 2 (Đồ thị song liên thông)**.** *1 đồ thị vô hướng $G = (V, E)$ được gọi là* song liên thông (biconnected) *nếu nó liên thông & không có đỉnh khớp, i.e., nếu xóa 1 đỉnh bất kỳ thì đồ thị vẫn liên thông.*

Intuition. Có vài cách mường tượng để hiểu nom na khái niệm đồ thị song liên thông:

1. Giống như 1 bản đồ gồm các thành phố bay lơ lửng trên không trung, tương ứng với các đỉnh của đồ thị, được nối nhau bởi các tuyến đường. Nếu 1 thành phố bị phá hủy thì đường đi có 1 trong 2 đầu mút đó cũng bị phá hủy theo (vì các thành phố bay lơ lửng trên không trung). Khi đó tính song liên thông có nghĩa là có thể di chuyển qua lại giữa các thành phố, & nếu bất cứ thành phố nào bị phá hủy, thì vẫn còn đường khác đường khác để đi từ 1 thành phố bất kỳ sang 1 thành phố khác. "Không đi cửa trước thì ta đi cửa sau" ý là vậy.

Định nghĩa 3 (Thành phần song liên thông)**.** *Cho $G = (V, E)$ là 1 đồ thị vô hướng. 1* thành song song liên thông *là 1 đồ thị con song liên thông cực đại của G.*

Cụ thể hơn, G' là 1 thành phần song liên thông của G thì:

1. G' là đồ thị con của G.

2. G' song liên thông.

3. G' là cực đại (maximal), i.e., không thể thêm đỉnh vào G' mà vẫn giữ được tính song liên thông.

Lemma 1. *Cho $G = (V, E)$ là 1 đồ thị vô hướng. 1 đỉnh $v \in V$ có thể thuộc vào nhiều thành phần liên thông khác nhau & các đỉnh thuộc nhiều thành phần song liên thông đều là khớp.*

Chứng minh. +++ □

3.1 Some algorithms to find biconnected components – Vài thuật toán tìm thành phần song liên thông

Có nhiều cách để tìm thành phần song liên thông, cách phổ biến nhất là duyệt theo kiểu Tarjan, cùng với phương pháp *nén đường* (path compression), & cấu trúc *các tập rời rạc* (disjoin sets) để tìm thành phần song liên thông.

3.1.1 Path compression – Phương pháp nén đường

Lemma 2. *Cho $G = (V, E)$ là 1 đồ thị vô hướng. Các đỉnh cùng nằm trên 1 chu trình đơn trong G sẽ cùng thuộc 1 thành phần song liên thông.*

Chứng minh. +++ □

Ý tưởng của phương pháp nén đường. Kết hợp DFS với tìm thành phần song liên thông: các đỉnh cùng thuộc 1 thành phần song liên thông sẽ được hợp nhất lại bằng disjoint-sets. Mỗi lần tìm thấy 1 chu trình có dạng $v_1, v_2, \ldots, v_n, v_1$, $v_i \neq v_j, \forall i, j \in [n]$, thì chu trình này tạo nên 1 thành phần song liên thông, ta sẽ hợp nhất các đỉnh này lại. Mỗi thành phần song liên thông sau khi hợp nhất được đại diện bởi đỉnh có bậc nhỏ nhất trên cây DFS, tuy nhiên, do đỉnh v có thể thuộc 1 hay vài thành phần song

liên thông khác nên ta chỉ hợp nhất các đỉnh của 1 thành phần liên thông mà không phải là đỉnh khớp, i.e., mỗi thành phần song liên thông $G' = (V', E')$, $V' \subset V$, của G được mã hóa (encoding) bởi:

$$\min_{v \in V',\ v \text{ is a not cut vertex of } G} \deg_{\text{DFS}} v.$$

Implementation – Cài đặt. Trong quá trình duyệt DFS, ta sẽ quản lý 1 stack chứa các đỉnh đang được duyệt, với các đỉnh đã được hợp nhất thì chỉ lưu đỉnh có bậc nhỏ nhất, & ta cũng cần lưu đỉnh con đang được duyệt của mỗi đỉnh.

```
#define N 30001
int root[N];
bool visited[N];
int active[N];
vector<int> stack;

// disjoint sets structure
int find(int u) {
    if (root[u] != u) root[u] = find(root[u]);
    return root[u];
}

void dfs(int u, const vector<vector<int>>& adj_list) {
    visited[u] = true;
    root[u] = u; // init
    stack.push_back(u);
    for (int v : adj_list[u])
    if (visited[v]) {
        v = find(active[v]);
        while (stack.back() != v) {
            root[find(stack.back())] = v;
            stack.pop_back();
        }
    }
    for (int v : adj_list[u])
    if (!visited[v]) {
        active[u] = v;
        dfs(v, adj_list);
    }
    if (stack.back() == u) stack.pop_back();
}
```

Bài toán 1 (SPOJ/safenet2 – mạng máy tính an toàn)**.** *Có $n \in \mathbb{N}^\star$ máy tính đánh số $1, 2, \ldots, n$, & $m \in \mathbb{N}$ dây cáp mạng, giữa 2 máy tính có thể có 1 hoặc nhiều đường dây cáp mạng nối chúng, không có cáp mạng nối 1 máy với chính nó. 2 máy tính có thể truyền dữ liệu cho nhau nếu có đường cáp nối trực tiếp giữa chúng hoặc truyền qua 1 số máy trung gian. 1 tập S các máy tính được gọi là* hệ thống an toàn *nếu dù 1 máy tính bất kỳ bị tấn công, e.g., do sự tò mò của người dân cứ thích truy cập & hack các trang cấm, thì trong số các máy tính còn lại, các máy thuộc tập S vẫn có thể truyền được dữ liệu cho nhau. Xác định số lượng lớn nhất có thể các máy tính của tập S.*

Input. *Dòng 1 chứ 2 số nguyên, $n \in [3 \cdot 10^4], m \in \overline{0, 10^5}$. m dòng tiếp theo ghi thông tin về các dây cáp mạng, gồm 2 chỉ số của 2 máy được dây đó nối trực tiếp.*

Output. *Ghi số nguyên duy nhất là số lượng máy tính lớn nhất tìm được.*

Sample.

safenet2.inp	safenet2.out
8 10 1 2 2 3 3 1 1 4 4 5 5 1 1 6 6 7 7 8 8 1	4

4 Miscellaneous