Disjoint Set Union (DSU) – Hợp Tập Hợp Rời Rạc

Nguyễn Quản Bá Hồng*

Ngày 22 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:

- *Disjoint Set Union (DSU) – Hợp Tập Hợp Rời Rạc.*
  PDF: URL: .pdf.
  TEX: URL: .tex.
- .
  PDF: URL: .pdf.
  TEX: URL: .tex.

Mục lục

1 Introduction to Disjoint Set Union – Nhập Môn Hợp Tập Hợp Rời Rạc

Resources – Tài nguyên.

1. Algorithms for Competitive Programming/disjoint set union.
2. Benjamin Qi, Andrew Wang, Nathan Gong, Michael Cao. USACO Guide/Disjoint Set Union.

   Abstract. The Disjoint Set Union (DSU) data structure, which allows you to add edges to a graph & test whether 2 vertices of a graph are connected.

   – Cấu trúc dữ liệu Disjoint Set Union (DSU), cho phép bạn thêm các cạnh vào đồ thị & kiểm tra xem 2 đỉnh của đồ thị có được kết nối hay không.
3. Ngô Quang Nhật. VNOI Wiki/Disjoint Set Union.
4. Wikipedia/disjoint-set data structure.

Disjoint Set Union (abbr., DSU) là 1 cấu trúc dữ liệu hữu dụng, thường xuất hiện trong các kỳ thi Lập trình Thi Đấu, & có thể được dùng để quản lý 1 cách hiệu 1 tập hợp của các tập hợp.

Bài toán 1. *Cho 1 đồ thị $G = (V, E)$ có $|V| = n \in \mathbb{N}^\star$ đỉnh, ban đầu không có cạnh nào, $E = \emptyset$. Ta cần xử lý 2 loại truy vấn:*

1. *Thêm 1 cạnh giữa 2 đỉnh $x, y \in V$ trong đồ thị, i.e., $E = E \cup \{\{x, y\}\}$ nếu G là đồ thị vô hướng & $E = E \cup \{(x, y)\}$ nếu G là đồ thị có hướng (trước tiên ta chỉ xét trường hợp đồ thị vô hướng cho đơn giản).*
2. *In ra* `yes` *nếu như 2 đỉnh x, y nằm trong cùng 1 thành phần liên thông. In ra* `no` *nếu ngược lại.*

*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.

1.1 Data Structure Disjoint Set Union – Cấu trúc dữ liệu Disjoint Set Union

Cho tiện, ta đánh số n đỉnh của đồ thị G bởi $1, 2, \ldots, n$ (trường hợp n đỉnh được dán nhãn bởi $v_1, v_2, \ldots, v_n$ hoàn toàn tương tự vì ta có thể làm việc trên chỉ số i của v_i), khi đó $V = [n]$. Giả sử G có $c \coloneqq$ `num_connected_component` $\in \mathbb{N}^\star$ (số thành phần liên thông) $C_1, C_2, \ldots, C_c$ với $\{C_i\}_{i=1}^{c}$ là 1 phân hoạch của $V = [n]$, i.e.:

$$\bigcup_{i=1}^{c} C_i = [n],\ C_i \cap C_j = \emptyset,\ \forall i, j \in [c],\ i \neq j.$$

Nếu ta coi mỗi đỉnh của đồ thị $G = (V, E)$ là 1 phần tử & mỗi thành phần liên thông (connected component) trong đồ thị là 1 tập hợp, truy vấn 1 sẽ trở thành gộp 2 tập hợp lần lượt chứa phần tử x, y thành 1 tập hợp mới & truy vấn 2 trở thành việc hỏi 2 phần tử x, y có nằm trong cùng 1 tập hợp không.

Để tiện tính toán & lý luận về mặt toán học cho riêng cấu trúc dữ liệu DSU, sau đây là 1 định nghĩa lai Toán–Tin mang tính cá nhân của tác giả [NQBH], hoàn toàn không chính thống trong Lý thuyết Đồ thị:

Định nghĩa 1 (Chỉ số thành phần liên thông)**.** *Cho đồ thị vô hướng $G = (V, E)$ với $V = [n]$. Gọi $C(i) \subset [n]$ là thành phần liên thông của $G = (V, E)$ chứa đỉnh $i \in [n]$ & gọi chỉ số của thành phần liên thông chứa đỉnh i là* $\mathrm{cid}(i)$*, i.e., $i \in C_{\mathrm{cid}(i)} \equiv C(i)$, với hàm* $\mathrm{cid} : [n] \to [c]$ *được gọi là* hàm chỉ số liên thông*.*

Với định nghĩa 1, ta có ngay

$$\begin{cases} i \in C(i) = C_{\mathrm{cid}(i)} \subset [n], \\ i, j \text{ are connected},\ \forall j \in C(i). \end{cases} \quad \forall i \in [n].$$

Ở đây, ta coi mỗi đỉnh của đồ thị tự liên thông với chính nó theo nghĩa đỉnh đó đến được (reachability) chính đỉnh đó thông qua 1 đường đi có độ dài 0, được gọi là 1 *đường đi tầm thường*, chứ không phải theo nghĩa khuyên (loop).

Lemma 1 (A characterization of connectedness – 1 đặc trưng hóa của tính liên thông)**.** *Cho đồ thị vô hướng $G = (V, E)$. (i) 2 đỉnh trên 1 đồ thị G không liên thông với nhau, i.e., không có đường đi nào trên G nối chúng khi & chỉ khi chúng thuộc 2 thành phần liên thông khác nhau, i.e.,*

$$i, j \text{ are not connected} \Leftrightarrow C(i) \neq C(j) \Leftrightarrow C(i) \cap C(j) = \emptyset \Leftrightarrow \mathrm{cid}(i) \neq \mathrm{cid}(j),\ \forall i, j \in [n].$$

(ii) 2 đỉnh trên đồ thị G liên thông với nhau, i.e., có đường đi trên trên G nối chúng khi & chỉ khi chúng cùng thuộc 1 thành phần liên thông, i.e.:

$$i, j \text{ are connected} \Leftrightarrow C(i) = C(j) \Leftrightarrow C(i) \cap C(j) \neq \emptyset \Leftrightarrow \mathrm{cid}(i) = \mathrm{cid}(j),\ \forall i, j \in [n].$$

Với truy vấn 1, giả sử 2 đỉnh $i, j \in [n]$ chưa có cạnh nối chúng trực tiếp, i.e., $\{i, j\} \notin E$. Có 2 trường hợp xảy ra:

- Trường hợp 1: Giả sử i, j cùng thuộc 1 thành phần liên thông, theo Lem. 1, có $C(i) = C(j)$, $\mathrm{cid}(i) = \mathrm{cid}(j)$ nên ta chỉ cần thêm cạnh $\{i, j\}$ vào tập cạnh E: $E \leftarrow E \cup \{\{i, j\}\}$ hay `edge_list.append({i, j})` mà không cần cập nhật c tập liên thông $\{C_i\}_{i=1}^{c}$ hay hàm chỉ số liên thông $\mathrm{cid}(\cdot)$.
- Trường hợp 2: Giả sử i, j thuộc 2 thành phần liên thông khác nhau, theo 1, có $C(i) \neq C(j)$, $\mathrm{cid}(i) \neq \mathrm{cid}(j)$. Khi đó, việc nối cạnh i, j với nhau, i.e., cập nhập tập cạnh bằng cách thêm cạnh $\{i, j\}$ vào tập cạnh E: $E \leftarrow E \cup \{\{i, j\}\}$ hay `edge_list.append({i, j})`, nhưng điều khác biệt ở đây nằm ở chỗ: ta cũng cần phải cập nhật tập các thành phần liên thông & hàm chỉ số liên thông như sau:
  1. Cập nhật thành phần liên thông chứa cả i & j bằng cách lấy hợp của 2 tập hợp $C(i), C(j)$, i.e., $C_{\mathrm{new}}(i) = C_{\mathrm{new}}(j) \coloneqq C(i) \cup C(j)$.
  2. Cập nhật hàm chỉ số liên thông: $\mathrm{cid}_{\mathrm{new}}(i) = \mathrm{cid}_{\mathrm{new}}(j) = \min\{\mathrm{cid}(i), \mathrm{cid}(j)\}$ (cũng có thể lấy max thay vì min nhưng theo quy ước của DSU data structure, ta nên lấy min để đảm bảo tính nhất quán[1]) & sau đó ta có thể đánh chỉ số các thành phần liên thông lại nếu cần vì sau khi nối i, j với nhau, số thành phần liên thông c đã giảm đi 1, i.e., $c \leftarrow c - 1$ hay `-c`.

Với truy vấn 2, với 2 đỉnh $i, j \in [n]$, $i \neq j$, ta có thể kiểm tra 2 đỉnh này có cùng nằm trong 1 thành phần liên thông hay không bằng cách so sánh $C(i)$ & $C(j)$ hoặc $\mathrm{cid}(i)$ & $\mathrm{cid}(j)$, nhờ Lem. 1. Việc thực hiện truy vấn này khá hiển nhiên do bản chất của cấu trúc dữ liệu DSU chính là rã sự liên thông thành các thành phần liên khác nhau để tiện quản lý.

Về mặt cài đặt, để giải Bài toán 1, ta sẽ xây dựng 1 cấu trúc dữ liệu có 3 thao tác sau:

1. `make_set(v)` tạo ra 1 tập hợp mới chỉ chứa phần tử v: `make_set(v)` $= \{v\}$.
2. `union_sets(a,b)` gộp tập hợp chứa phần tử a & tập hợp chứa phần tử b thành 1, i.e., bước cập nhật thành phần liên thông chung bằng cách lấy hợp của 2 thành phần liên thông tương ứng: $C_{\mathrm{new}}(i) = C_{\mathrm{new}}(j) \coloneqq C(i) \cup C(j)$ đã để cập.

[1] Consistency is 1 of the kings in natural sciences.

3. `find_set(v)` cho biết *đại diện* (representative) của tập hợp có chứa phần tử v (phần tử đại diện cho v không nhất thiết phải là v, e.g., người đỡ đầu cho 1 tập hợp các đứa trẻ trong 1 trại trẻ mồ côi). Đại diện này là sẽ 1 phần tử của tập hợp đó & có thể thay đổi sau mỗi lần gọi thao tác `union_sets` (e.g., nếu chọn phần tử đại diện là đỉnh có giá trị nhỏ nhất thì nếu tìm được đỉnh mới nhỏ hơn phần tử đại diện hiện tại, thì phải cập nhật phần tử đại diện mới này). Ta có thể sử dụng đại diện đó để kiểm tra 2 phần tử có nằm trong cùng 1 tập hợp hay không. a, b nằm trong cùng 1 tập hợp nếu như đại diện của 2 tập chứa chúng là giống nhau & không nằm trong cùng 1 tập hợp nếu ngược lại.

Ta có thể xử lý 3 thao tác này 1 cách hiệu quả với các tập hợp được biểu diễn dưới dạng các cây (tree-based representation), mỗi phần tử là 1 đỉnh & mỗi cây tương ứng với 1 tập hợp. Gốc của mỗi cây sẽ là đại diện của tập hợp đó.

2 Miscellaneous