CHUYÊN ĐỀ: LÝ THUYẾT ĐỒ THỊ

Đặng Phúc An Khang*

Ngày 8 tháng 7 năm 2025

Tóm tắt nội dung

Code:

- C/C++: https://github.com/GrootTheDeveloper/OLP-ICPC/tree/master/2025/C%2B%2B.
- Python:

Tài khoản trên các Online Judge:

- Codeforces: https://codeforces.com/profile/vuivethoima.
- VNOI: oj.vnoi.info/user/Groot.
- IUHCoder: oj.iuhcoder.com/user/ankhang2111.
- MarisaOJ: https://marisaoj.com/user/grootsiuvip/submissions.
- CSES: https://cses.fi/user/212174.
- UMTOJ: sot.umtoj.edu.vn/user/grootsiuvip.
- SPOJ: www.spoj.com/users/grootsiuvip/.
- POJ: http://poj.org/userstatus?user_id=vuivethoima.
- ATCoder: https://atcoder.jp/users/grootsiuvip
- OnlineJudge.org: vuivethoima
- updating...

Mục lục

*E-mail: ankhangluonvuituoi@gmail.com. Tây Ninh, Việt Nam.

1 Preliminaries – Kiến thức chuẩn bị

Resources – Tài nguyên.

1. [**CP10**]. *CP10. Competitive Programming* https://drive.google.com/drive/folders/1MTEVHT-7nBnMJ7C9LgyAR_pEVSE3FlKz?fbclid=IwAR3TovIj2rKCRe1a4oZxW-LQCoEoVkipVAvCzwrr0nJ6GzcAd47P6LO1Rwc
2. [**cp-algorithms**]. *Algorithms for Competitive Programming* https://cp-algorithms.com
3. [**VNOI-WIKI**]. *Thư viện VNOI* https://wiki.vnoi.info

2 Kiến thức

2.1 Giới thiệu về đồ thị và thuật toán DFS

2.1.1 Lý thuyết đồ thị là gì?

Lý thuyết đồ thị là một nhánh của toán học, cụ thể thuộc toán rời rạc. Lý thuyết đồ thị chuyên nghiên cứu các bài toán liên quan đến việc biểu diễn và phân tích các sự vật, hiện tượng hoặc trạng thái có mối quan hệ lẫn nhau thông qua mô hình đồ thị.

Một số minh họa điển hình: Mạng lưới giao thông, cây phả hệ (cây gia phả), mạng máy tính, sơ đồ tổ chức, v.v.

2.1.2 Một số khái niệm căn bản trong lý thuyết đồ thị

1. **Đỉnh:** Được biểu diễn nhằm mục đích thể hiện sự vật, sự việc hay một trạng thái.
2. **Cạnh:** Biểu diễn cho mối quan hệ giữa 2 đỉnh với nhau. **Lưu ý:** Giữa 2 đỉnh trong đồ thị có thể có cạnh, không có, hoặc có thể có nhiều cạnh với nhau. Cạnh được chia thành 2 dạng:
   (a) **Cạnh vô hướng:** Nếu một cạnh vô hướng nối 2 đỉnh u và v, thì u có thể đến v trực tiếp vầ ngược lại.
   (b) **Cạnh có hướng:** Nếu một cạnh có hướng nối từ đỉnh u đến đỉnh v, thì ta có thể đi trực tiếp từ u đến v, nhưng không thể đi ngược lại từ v đến u trừ khi có một cạnh khác từ v đến u.
3. **Đường đi:** Một đường đi là một danh sách các đỉnh $x_1, x_2, x_3, x_4, \ldots, x_k$. Trong đó 2 đỉnh x_i và x_{i+1} thì có một đường nối trực tiếp để đi từ $x_i \to x_{i+1}$.
4. **Trọng số:** Là một giá trị trên cạnh (hoặc trên đỉnh) nhằm thể hiện một thông số nào đó với bài toán ta đang xét.

2.1.3 Thuật toán DFS

Thuật toán DFS (Depth-First Search – Duyệt theo chiều sâu) là một trong những thuật toán cơ bản để duyệt hoặc tìm kiếm trên đồ thị. Ý tưởng chính là xuất phát từ một đỉnh ban đầu, đi sâu theo từng nhánh con của đồ thị cho đến khi không còn đỉnh nào có thể đi tiếp, sau đó quay lui để khám phá các nhánh khác.
DFS có thể được cài đặt đệ quy hoặc sử dụng ngăn xếp. Nó thường được dùng để:

- Kiểm tra tính liên thông của đồ thị
- Tìm thành phần liên thông
- Phát hiện chu trình
- Tìm đường đi trong mê cung hoặc đồ thị

2.1.4 Ý tưởng cài đặt thuật toán DFS

DFS thường được cài đặt bằng đệ quy hoặc sử dụng ngăn xếp. Trong cài đặt đệ quy, ta cần một mảng đánh dấu để theo dõi các đỉnh đã được thăm nhằm tránh lặp vô hạn trong trường hợp đồ thị có chu trình.
Các bước cơ bản trong cài đặt DFS đệ quy:

1. Khởi tạo một mảng `visited[]` để đánh dấu các đỉnh đã được duyệt, với ý nghĩa: `visited[u] = true / false` nếu đỉnh u đã thăm / chưa thăm.
2. Gọi hàm `DFS(u)` tại đỉnh bắt đầu u.
3. Trong mỗi lần gọi:
   - Đánh dấu `visited[u] = true`.
   - Duyệt qua tất cả các đỉnh kề v của u:
     - Nếu v chưa được thăm (`visited[v] == false`), đệ quy gọi `DFS(v)`.

2.1.5 Cài đặt DFS sử dụng đệ quy trong C++

Listing 1: Thuật toán DFS sử dụng đệ quy

```cpp
#include <iostream>
#include <vector>
using namespace std;

const int MAXN = 100005; // So dinh toi da
vector<int> adj[MAXN];   // Danh sach ke
bool visited[MAXN];      // Mang danh dau

void DFS(int u) {
    visited[u] = true;
    cout << "Tham dinh: " << u << endl;
    for (int v : adj[u]) {
        if (!visited[v]) {
            DFS(v);
        }
    }
}
int main() {
    int n, m; // so dinh va so canh
    cin >> n >> m;
    for (int i = 0; i < m; i++) {
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u); // Neu la do thi vo huong
    }
    for (int i = 1; i <= n; i++) {
        visited[i] = false;
    }
    // Goi DFS tu dinh 1 (hoac 1 dinh bat ky)
    DFS(1);

    return 0;
}
```

Độ phức tạp: $O(V + E)$ với V là số đỉnh, E là số cạnh.

2.2 Topological Sorting

2.3 Khớp và Cầu (Joints and Brides)

2.4 Thành phần liên thông mạnh (Strongly Connected Components)

2.5 Thuật toán BFS

2.6 Thuật toán Dijkstra + Heap

2.7 Disjoint Set Union (DSU)

2.8 Maximum Flow and Maximum Matching

2.9 Minimum Cut

2.10 Euler Tour

2.11 Lowest Common Ancestor

2.12 Heavy Light Decomposition

2.13 Centroid Decomposition

2.14 2-SAT

3 Miscellaneous

3.1 Contributors