💹

Photo by Jigar Panchal on Unsplash

Thành phần liên thông trong đồ thị

Note

I. Giải thích ngắn gọn tiêu đề

Thành phần liên thông trong lý thuyết đồ thị là một khái niệm quan trọng và có nhiều ứng dụng trong thực tế, nhờ vào khái niệm này mà chúng ta có các lĩnh vực mới cho ngành Khoa học máy tính hiện đại như là học máy với dữ liệu đồ thị hoặc thị giác máy tính 3 chiều.

Để tránh hiểu lầm về vấn đề xác định thành phần liên thông mạnh với đồ thị có hướng thì trong phạm vi bài viết chỉ xét các thành phần liên thông trong một đồ thị vô hướng.

II. Ứng dụng của thành phần liên thông trong đồ thị

1. **Mạng xã hội:**
    - Trong các mạng xã hội, ta có thể hiểu thành phần liên thông là dùng để biểu diễn các nhóm người dùng hoặc là cộng đồng có sự liên quan chặt chẽ với nhau.
    - Phân tích các thành phần liên thông này giúp hiểu biết thêm về cấu trúc mạng và mối quan hệ giữa các tập hợp người dùng.
2. **Internet và Mạng máy tính:**
    - Trong mạng Internet, thành phần liên thông có thể đại diện cho các trang web hoặc máy chủ có thể tiếp cận trực tiếp hoặc gián tiếp thông qua các liên kết.
    - Quản lý và tối ưu hóa các kết nối trong mạng máy tính dựa trên thành phần liên thông.

3. **Dự báo bệnh dịch và dịch tễ học:**
    - Trong nghiên cứu về bệnh dịch, việc hiểu về thành phần liên thông giữa các cộng đồng có thể giúp dự đoán và kiểm soát sự lây lan của bệnh.
4. **Tìm kiếm và truy vết:**
    - Trong các hệ thống tìm kiếm, thành phần liên thông giúp tìm kiếm hiệu quả giữa các trang web hoặc tài nguyên.
    - Các thuật toán tìm kiếm có thể tận dụng cấu trúc thành phần liên thông để cải thiện thời gian tìm kiếm.
5. **Tối ưu hóa mạng giao thông:**
    - Trong các hệ thống tìm kiếm, thành phần liên thông giúp tìm kiếm hiệu quả giữa các trang web hoặc tài nguyên.
    - Các thuật toán tìm kiếm có thể tận dụng cấu trúc thành phần liên thông để cải thiện thời gian tìm kiếm.
6. **Mô hình hóa các hệ thống:**
    - Thành phần liên thông thường được sử dụng để mô hình hóa và phân loại các phần của một hệ thống phức tạp thành các thành phần riêng biệt có thể nghiên cứu và quản lý một cách hiệu quả.
7. **Quy hoạch tài nguyên mạng:**
    - Trong quản lý tài nguyên mạng, việc hiểu cấu trúc thành phần liên thông có thể giúp quyết định về vị trí lưu trữ dữ liệu và tài nguyên để tối ưu hóa hiệu suất.
8. **Mô phỏng và nghiên cứu khoa học:**
    - Trong lĩnh vực nghiên cứu và mô phỏng, thành phần liên thông giúp tạo ra các mô hình đơn giản và dễ quản lý của hệ thống phức tạp.
9. **Con đường mới cho thị giác máy t**ính:
    - Thuật toán CCL và CCA chính là một trong các ứng dụng của thành phần liên thông nói riêng và lý thuyết đồ thị nói chung vào trong thị giác máy tính với ý tưởng lớn là các tập con của các thành phần liên thông được gán nhãn duy nhất dựa và một heuristic cụ thể cho trước. Là sự phân biệt rõ ràng với bài toán Segmentation.

III. Mối quan hệ trong đồ thị liên thông

Một cách khác để định nghĩa ***thành phần liên thông*** là thông qua việc sử dụng các lớp **tương đương** của **quan hệ tương đương** trên tập hợp các đỉnh của đồ thị. Trong đồ thị vô hướng, đỉnh v được coi là tới được từ đỉnh u nếu có đường đi từ u đến v, bao gồm cả trường hợp đường đi có độ dài 0 từ một đỉnh đến chính nó và có thể xuất hiện nhiều lần trên đường đi. Quan hệ "tới được" được coi là một quan hệ tương đương.

Từ đó ta có được các tính chất khi xét một đồ thị liên thông như sau:

- Tính chất phản xạ: luôn có đường đi độ dài 0 từ một đỉnh đến chính nó.
- Tính chất đối xứng: nếu có đường từ *u* tới *v* thì cũng có đường từ *v* tới *u*.
- Tính chất bắc cầu: nếu có đường từ u tới v và đường từ v tới w thì khi nối hai đường này, ta có đường từ u tới w.

IV. Các thuật toán

1. ***Gán nhãn thành phần liên thông:*** theo các đơn giản nhất của Khoa học máy tính cổ điển ta có thể dùng thuật toán gán nhãn các thành phần liên thông trong đồ thị với ý tưởng lớn là với mỗi đỉnh chưa được gán nhãn vào một thành phần liên thông, ta tiến hành gán nhãn và tiến hành gọi đệ quy xuống 1 hàm con để tìm kiếm, trong hàm con này ta thực hiện việc gán nhãn dựa vào nhãn cha, nếu nhãn cha và con không trùng thì ta tiến hành gán nhãn vào sau đó lặp lại quá trình này cho tới hết các đỉnh trong đồ thị, việc lặp lại được thực hiện thằng kỹ thuật đệ quy.
2. ***DFS:*** một cách ứng dụng khác của thuật toán DFS là ta sẽ duyệt theo chiều sâu toàn bộ các đỉnh trong đồ thị và mỗi lần duyệt ta xét các đỉnh kề với đỉnh đang duyệt và đánh dấu lại các đỉnh này, quá trình này được lặp đi lặp lại cho tới khi không thể tìm kiếm thêm được thì dừng lại việc duyệt chiều sâu.
3. ***BFS:*** Tương tự với DFS ta cũng có thể duyệt đồ thị theo chiều rộng để thực hiện việc liệt kê các thành phần liên thông trong đồ thị với ý tưởng tương tự BFS nhưng thay vì theo ưu tiên theo thứ tự sâu thì ta ưu tiên theo thứ tự rộng.

> 💡 Trong đa số bài toán ta sẽ dùng DFS để thực hiện vì thuật toán DFS mang lại sự cô đọng và xúc tích cho việc cài đặt, BFS và Gán nhãn có phần "nhập nhằng" hơn khi BFS phải dùng dữ liệu hàng đợi hỗ trợ cho việc duyệt đồ thị còn Gán nhãn thì lại dùng đệ quy phi tuyến lồng nhau làm mã nguồn có phần khó tiếp cận.

V. Cài đặt

Phạm vi cài đặt: trong phạm vi bài viết này, ta sẽ chỉ xét một bài toán cụ thể trên một tập đồ thị vô hướng và hữu hạn với đầu vào cho trước.

Ngôn ngữ lập trình: sử dụng ngôn ngữ C++ để tiến hành cài đặt vì cú pháp chặt chẽ, bộ thư viện tiêu chuẩn kèm theo đầy mạnh mẽ trong việc cài đặt các chương trình cần tới cấu trúc dữ liệu để hỗ trợ.

Mô tả bài toán: ta sẽ nhận đầu vào là một đồ thị với số đỉnh và số cạnh được xác định từ trước, mục tiêu cụ thể là **in ra các thành phần liên thông trong đồ thị đã được liệt kê theo thứ tự tăng dần theo chỉ số đỉnh.**

Link đề bài cụ thể: http://csloj.ddns.net/problem/538

1. Cài đặt bằng Gán nhãn:

```C++
#include "bits/stdc++.h"
using namespace std;

const int maxn = 1e6 + 9;
int n, m, a[1009][1009], u, v, vertex[maxn];

inline void findCompcon(int n, int label[], int i) {
    for (int j = 0; j < n; ++j)
        if (a[i][j] != 0 && label[j] != label[i]) {
            label[j] = label[i];
            findCompcon(n, label, j);
        }
}

inline void checkCompcon() {
    int label[1009], i, SothanhphanLT = 0;
    fill(label, label + n, 0);
    for (i = 0; i < n; ++i)
        if (label[i] == 0) {
            ++SothanhphanLT;
            label[i] = SothanhphanLT;
            findCompcon(n, label, i);
        }
    int z = 0, x = 0;
    for (i = 1; i <= SothanhphanLT; ++i) {
        int cnt = 0, f = -1;
        for (int j = 0; j < n; ++j) {
            if (label[j] == i)
                ++cnt;
            f = max(cnt, f);
        }
        vertex[z] = f;
        ++z;
    }
    cout << SothanhphanLT << '\n';
    for (i = 1; i <= SothanhphanLT; ++i) {
        cout << vertex[x] << ' ';
        for (int j = 0; j < n; ++j)
            if (label[j] == i)
                cout << j + 1 << ' ';
        ++x;
        cout << '\n';
    }
}
```

```
int32_t main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
    cin >> n >> m;
    for (int i = 0; i < m; i++) {
        cin >> u >> v;
        --u, --v, a[u][v] = 1, a[v][u] = 1;
    }
    checkCompcon();
}
```

2. Cài đặt bằng BFS:

- Đối với BFS, ta bắt buộc phải dùng hàng đợi để hỗ trợ quá trình duyệt đồ thị, các cách cài đặt hàng đợi có thể dùng danh sách liên kết hoặc mảng 1 chiều để giải quyết vấn đề nhưng trong bài giải sẽ dùng hàng đợi từ thư viện tiêu chuẩn của C++ để tránh việc nhập nhằng trong vấn đề cài đặt cấu trúc dữ liệu mà ta chỉ tập trung vào thuật toán.

```
C++
#include "bits/stdc++.h"
using namespace std;

const int maxN = 1e5 + 9;
bool check[maxN];
int n, m, ans;
vector<vector<int>> a, res;

void bfs(int s) {
    queue<int> q;
    check[s] = true;
    q.push(s);
    while (!q.empty()) {
        int u = q.front();
        auto it = lower_bound(res[ans].begin(), res[ans].end(), u);
        res[ans].insert(it, u);
        q.pop();
        for (auto x : a[u]) {
            if (!check[x]) {
                check[x] = true;
                q.push(x);
            }
        }
    }
}
```

```
void findComponent() {
    for (int i = 1; i <= n; ++i) {
        if (!check[i]) {
            ++ans;
            bfs(i);
        }
    }
    cout << ans << '\n';
    for (int i = 1; i <= ans; ++i) {
        cout << res[i].size() << ' ';
        for (auto x : res[i]) {
            cout << x << ' ';
        }
        cout << '\n';
    }
}

int32_t main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
    cin >> n >> m;
    a = res = vector<vector<int>>(n + 1);
    for (int i = 1, u, v; i <= m; ++i) {
        cin >> u >> v;
        a[u].emplace_back(v);
        a[v].emplace_back(u);
    }
    findComponent();
}
```

3. Cài đặt bằng DFS:

C++ Wrap Copy

```
#include "bits/stdc++.h"
using namespace std;

const int maxN = 1e5 + 9;
bool check[maxN];
int n, m, ans;
vector<vector<int>> a, res;

void dfs(int u) {
    check[u] = true;
    auto it = lower_bound(res[ans].begin(), res[ans].end(), u);
    res[ans].insert(it, u);
    for (int x : a[u]) {
```

```
        if (!check[x]) {
            dfs(x);
        }
    }
}

void findComponent() {
    for (int i = 1; i <= n; ++i) {
        if (!check[i]) {
            ++ans;
            dfs(i);
        }
    }
    cout << ans << '\n';
    for (int i = 1; i <= ans; ++i) {
        cout << res[i].size() << ' ';
        for (auto x : res[i]) {
            cout << x << ' ';
        }
        cout << '\n';
    }
}

int32_t main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
    cin >> n >> m;
    a = res = vector<vector<int>>(n + 1);
    for (int i = 1, u, v; i <= m; ++i) {
        cin >> u >> v;
        a[u].emplace_back(v);
        a[v].emplace_back(u);
    }
    findComponent();
}
```

Author: longdnk
Master student at University of Science, VNU-HCM, Vietnam
Tag: Mathematical Methods in Computer Science, Explainable AI, Deep Learning