Thuật toán Tarjan

Hoàng Văn Thiên

Tóm tắt nội dung

Thuật toán Tarjan có hai ứng dụng chính: Tìm khớp và cầu trong đồ thị vô hướng, và Tìm thành phần liên thông mạnh trong đồ thị có hướng.

1 Tìm khớp và cầu trong đồ thị vô hướng

Ví dụ mở đầu

Cho một đồ thị liên thông vô hướng $G = (V, E)$. Mỗi đỉnh và cạnh có thông số cho biết chi phí để phá hủy đối tượng đó. Tìm đỉnh hoặc cạnh để biến G thành đồ thị không liên thông.

1.1 Định nghĩa

- Đồ thị vô hướng liên thông khi và chỉ khi tồn tại đường đi giữa mọi cặp đỉnh.
- Thành phần liên thông của một đồ thị là một đồ thị con trong đó giữa bất kì hai đỉnh nào đều có đường đi đến nhau, và không thể nhận thêm bất kì một đỉnh nào mà vẫn duy trì tính chất trên
- Khớp: là đỉnh mà nếu ta bỏ nó (cùng các cạnh liên thuộc với nó) ra khỏi đồ thị, số thành phần liên thông của đồ thị sẽ tăng lên.
- Cầu: là cạnh mà nếu ta bỏ nó ra khỏi đồ thị, số thành phần liên thông của đồ thị sẽ tăng lên.
- Đồ thị song liên thông: Đồ thị không có khớp.
- Tổ tiên của một đỉnh có thể là cha của nó, hoặc là chính nó.
- Back edge: Cạnh đưa đỉnh hiện tại trong hàm DFS quay về một đỉnh đã được duyệt từ trước.

1.2 Ý tưởng

Một cách ngắn gọn, thuật toán Tarjan áp dụng DFS để hình thành cây DFS.

Trên cây DFS, đỉnh được thăm trước gọi là đỉnh cha.

Nếu con của một đỉnh **không có** đường đi (thông qua back edge hoặc/và dãy đỉnh con) về cha của đỉnh này, thì đỉnh này là khớp. Vì nếu bỏ đỉnh này đi, con của nó không thể đến cha của nó.

Ngoại lệ, đỉnh gốc của cây DFS là khớp nếu và chỉ nếu có từ hai con trực tiếp trở lên.

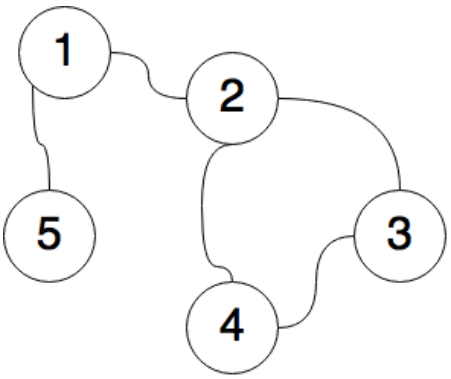
1.3 Thuật toán

Thuật toán Tarjan, theo nhận xét của John Edward Hopcroft và Robert Endre Tarjan, là một biến thể của DFS, nên độ phức tạp vẫn là $O(V + E)$.

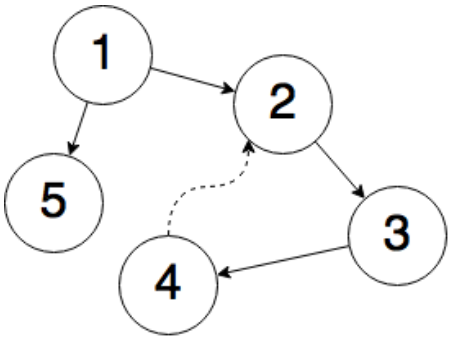
Chúng ta định nghĩa một số biến như sau:

- `num[u]`: Thứ tự duyệt của đỉnh `u` khi DFS.
- `low[u]`: Giá trị `num[u]` nhỏ nhất đạt được trong cây DFS con của đỉnh `u`.
  Ban đầu, `low[u] = num[u]`. Giá trị chỉ có thể giảm nếu duyệt được một chu kỳ có 3 đỉnh trở lên.
- `cnt`: Biến đếm thứ tự DFS của các đỉnh.

Trên cây DFS, nếu đỉnh gốc có hai con trở lên thì nó được tính là một khớp. Thuật toán Tarjan bỏ qua trường hợp đặc biệt này.



Hình 1: Đồ thị vô hướng



Hình 2: Trình tự duyệt DFS

Giải thuật DFS đỉnh `u` được tiến hành như sau:

1. Đặt `num[u] = low[u] = ++cnt`.
2. Với mỗi đỉnh `v` kề với `u`, ta chia ra hai trường hợp:
   - Nếu `v` chưa thăm:
     (a) Ghi nhận đỉnh `u` là cha của đỉnh `v` trên cây DFS.
     (b) Nếu đỉnh `u` là gốc của cây DFS, cập nhật số lượng con của nó.
     (c) `DFS(v)`
     (d) Nếu `low[v] >= num[u]` (từ `v` chỉ có thể đi đến một đỉnh từ `u` trở xuống trên cây DFS), thì `u` là khớp. Vì nếu không có `u` thì mọi đỉnh từ `v` trở xuống không đến được những đỉnh trên `u`.
     (e) Nếu `low[v] > num[u]` (từ `v` chỉ có thể đi đến một đỉnh dưới `u` nhưng không bao gồm `u` trên cây DFS) thì cạnh (u,v) là cầu.

(f) Tối thiểu giá trị `low[u]` theo giá trị `low[v]`.

- Nếu `v` đã thăm (chúng ta có một chu trình) và `v` không phải là cha của `u` (chu trình có hơn 2 đỉnh) thì tối thiểu `low[u]` theo `num[v]`.

```
int dfsCount = 0, root = 1, child = 0;
// ...
void tarjanAPB(int u) {
    num[u] = low[u] = ++dfsCount;
    for(int v : neighbor[u]) {
        if (num[v] == 0) {
            parent[v] = u;
            if (u == root) ++child;
            tarjanAPB(v);
            low[u] = min(low[u], low[v]);
            if (low[v] >= num[u]) AP[u] = true;
            if (low[v] > num[u]) bridge.push_back(make_pair(u,v));
        } else
            if (parent[u] != v) {
                low[u] = min(low[u], num[v]); } }
}
int main() {
    // ...
    for(int i = 1; i <= V; ++i)
        if (num[i] == 0) {
            root = i;
            child = 0;
            tarjanAPB(i);
            AP[i] = child > 1; } }

```

2 Tìm các thành phần liên thông mạnh trong đồ thị có hướng

Ví dụ mở đầu

Trong mạng lưới giao thông thành phố $G = (V, E)$ có các đường một chiều. Thành phố muốn rằng với hai vị trí x và y trong thành phố, luôn có đường từ x đến y và ngược lại. Hãy kiểm tra xem mạng lưới này đã thỏa mãn ý muốn của họ hay chưa.

2.1 Định nghĩa

Trong đồ thị có hướng, các định nghĩa sau cần phân biệt:

- Thành phần liên thông mạnh: đồ thị con tối đại có đường đi từ một đỉnh bất kỳ đến một đỉnh bất kỳ khác.

- Thành phần liên thông yếu: đồ thị con tối đại có đường đi giữa hai đỉnh bất kỳ trên đồ thị vô hướng nền.

Như vậy, với ví dụ mở đầu trên, ta cần kiểm tra xem đồ thị đã cho có đúng 1 thành phần liên thông hay không.

2.2 Ý tưởng

Trong quá trình thăm một đỉnh, sau khi đã thăm xong mỗi nhánh bằng DFS, ta truyền thông tin về đỉnh cha xa nhất đến được của hàm đệ quy vừa thực hiện cho đỉnh hiện tại.

Khi thăm hết các nhánh, ta kiểm tra xem ta đã thăm ngược lại đỉnh cha trong **các** hàm đệ quy vừa rồi hay chưa.

- Nếu chưa: ta đã tìm được một thành phần liên thông mạnh, bao gồm các đỉnh thuộc cây con của cây DFS tính từ điểm này trở xuống (**tất nhiên**, bỏ ra những đỉnh đã thuộc vào thành phần liên thông mạnh khác xét trước đó).
  Ngay lập tức, xóa bỏ thành phần này khỏi cây DFS hiện tại (cập nhật lại stack, `onstack`) để tránh bị nhầm lẫn qua lại giữa các thành phần liên thông mạnh.

2.3 Thuật toán

Độ phức tạp: $O(V + E)$

Chúng ta định nghĩa một số biến như sau:

- `vector<int> stk`: là stack lưu trữ thứ tự duyệt DFS các đỉnh.
- `bool onstack[.]`: mang giá trị true nếu đỉnh nằm trong stack và ngược lại.

Giải thuật thăm một đỉnh `u` như sau:

1. Thêm `u` vào stack, đánh dấu `onstack`, đánh số thứ tự DFS của đỉnh.
2. Với mọi đỉnh `v` kề với u:
   (a) Nếu đỉnh chưa từng được thăm (số thứ tự bằng 0): Đệ quy thăm `v`.
   (b) Nếu `onstack[v]`, tối thiểu `low[u]` theo `low[v]`.
3. Nếu `low[u] == num[u]` (không đi ngược lên đỉnh cha được → hoàn thành một thành phần liên thông mạnh):
   Khởi tạo một vùng nhớ cho thành phần liên thông mới (có thể dùng `vector`). Lần lượt xóa khỏi stack cho đến khi `u` bị lấy ra thì ngưng. Lưu ý rằng lấy đến đâu thì sửa `onstack` và thêm vào thành phần liên thông hiện tại.

```cpp
int dfsCount = 0, sccCount = 0;
vector<int> scc[V];
// ...
void tarjanSCC(int u) {
    stk.push_back(u);
    low[u] = num[u] = ++dfsCount;
    onstack[u] = true;
    for(int v : neighbor[u]) {
        if (num[v] == 0) tarjanSCC(v);
        if (onstack[v]) low[u] = min(low[u], low[v]);
    }
    if (num[u] == low[u]) {
        ++sccCount;
        while (1) {
            int v = stk.back();
            scc[sccCount].push_back(v);
            onstack[v] = false;
            stk.pop_back();
            if (u == v) break; } } }
int main() {
    // ...
    for(int i = 1; i <= V; ++i)
        if (num[i] == 0) tarjanSCC(i); }

```

— Hết —