KHAI THÁC KỸ THUẬT DUYỆT ĐỒ THỊ ƯU TIÊN CHIỀU SÂU (Depth First Search)

Lê Thanh Bình
THPT Nguyễn Trãi - Hải Dương

Kỹ thuật duyệt đồ thị ưu tiên chiều sâu (Depth First Search - DFS) là một trong những kỹ thuật thăm các đỉnh bắt đầu từ một đỉnh xuất phát hay được sử dụng trong các bài toán về lý thuyết đồ thị. Trong bài này, tôi không đi sâu vào kỹ thuật duyệt đồ thị ưu tiên chiều sâu mà tập trung vào việc khai thác các kết quả có được từ việc duyệt đồ thị ưu tiên chiều sâu, từ đó có được một cách nhìn tổng quát vê các dạng toán liên quan đến DFS.

I-Những kết quả chủ yêu của duyệt đồ thị ưu tiên chiều sâu (DFS)

Tôi sẽ không đi sâu vào code của DFS mà chỉ nhắc lại một số kết quả có được từ DFS, qua đó có một cách hình dung khái quát về đồ thị vô hướng cũng như đồ thị có hướng. Trên cơ sở hình dung này có được một cách nhìn tổng hợp về các bài toán liên quan đến DFS.

1) Cây duyệt chiều sâu DFS

Trong quá trình DFS, nếu với mỗi đỉnh u ta có đỉnh $prev[u]$ là số hiệu của đỉnh mà từ đó thủ tực DFS gọi đệ qui đến u. Xây dựng đồ thị con với các cạnh là $(prev[u], u)$ ta có được một cây. Cây này được gọi là cây DFS và các cạnh trên cây này thường được gọi là các cạnh liền. Cây DFS có thể coi là **"bộ khung"** của đồ thị, cho một hình dung đầu tiên về hình dáng của đồ thị.
Các cạnh còn lại thường được vẽ bằng "**nét đứt".**

2) Chỉ số DFS và chỉ số ngược DFS

Kết quả DFS còn có ba mảng quan trọng nữa:

- Mảng **num[1..maxn]**: Mảng cho biết thứ tự duyệt DFS của các đỉnh (thứ tự mà mỗi đỉnh bắt đầu duyệt)
- Mảng **low[1..maxn]**: Mảng thứ tự ngược. Với đỉnh u, $low[u]$ là thứ tự (giá trị num) nhỏ nhất có thể đi đến tược từ u bằng cách đi xuôi xuống theo các cạnh liền (các cạnh trên cây DFS) và đi ngược lên không quá một lần theo cạnh nét đứt)
- Mảng **tail[1..maxn]:** Mảng cho biết thời điểm kết thúc duyệt DFS của mỗi đỉnh

Việc có được hai mảng $num[\dots], low[\dots]$ cho phép phân loại được các cạnh và các đỉnh của đồ thị vô hướng.

3) Thứ tự topo của DFS

Trong thủ tục DFS mỗi lần gán $num[u] = id$ ta đặt $tp[id] = u$. Bằng cách này ta xây dựng được một mảng $tp[1], tp[2], \dots, tp[n]$ cho ta danh sách các đỉnh liệt kê theo thứ tự. Mảng này còn được gọi là mảng thứ tự topo DFS. Có một số điều đáng quan tâm trên mảng thứ tự topo này:

- Các đỉnh trên cây con gốc u của cây duyệt DFS (cây nét liền) là một đoạn liên tục trên mảng $tp[...]$ từ vị trí $num[u]$ đến vị trí $tail[u]$
- Đỉnh u là tổ tiên của v trên cây DFS khi và chỉ khi $num[u] \leq num[v] \leq tail[v] \leq tail[u]$

4) Giả lệnh mô tả code của DFS:

Ta tổng kết việc tính các mảng **prev[...], num[...], low[...], tail[...], tp[...]** bằng thủ tục được mổ tả như dưới đây:

```
procedure DFS(u);
begin
    cx[u]=false;
    inc(id); num[u]:=id; low[u]=id;
    tp[id]=u;
    for v ∈ ke(u) if cx[v] then
    begin
        prev[v]=u;
        DFS(v);
        if low[v]<low[u] then low[v]:=low[u];
    end else if num[v]<low[u] then low[u]:=num[v];
    tail[u]:=id;
end;
```

5) Thuật toán Tarjan

Thuật toán Tarjan là thuật toán tìm thành phần liên thông mạnh (có thể cải biên một chút để tìm thành phần song liên thông trên đồ thị vô hướng):

```
procedure Tarjan(u: longint);
begin
    cl[u]:=1;
    inc(id); num[u]:=id; low[u]:=id;
    inc(sn); s[sn]:=u;
    for v ∈ ke(u) if cl[v]=0 then
    begin
        Tarjan(v);
        low[u]=min(low[u],low[v]);
    end else if cl[v]=1 then low[u]=min(low[u],num[v]);
    if num[u]=low[u] then
    begin
        inc(slt);
        repeat
            v=s[sn]; dec(sn);
            ltm[v]:=slt; cl[v]:=2;
```

```
        until u=v;
end;
```

Một nhận xét rất quan trọng là số hiệu các thành phần liên thông mạnh (song liên thông) luôn cho ta một thứ tự topo DFS trên đồ thị mà mỗi đỉnh là một thành phần liên thông mạnh (song liên thông)

II-Các ứng dụng chính:

1) Cạnh cầu

Một cạnh được gọi là cạnh cầu trên đồ thị vô hướng nếu như bỏ cạnh này ra khỏi đồ thị thì số thành phần liên thông của đồ thị mới tăng lên. Dễ thấy rằng cạnh cầu của đồ thị không thể là cạnh nét đứt vì việc bỏ cạnh nét đứt sẽ không ảnh hưởng đến việc "đi lại" giữa các đỉnh (cây DFS đã duy trì điều này). Do vậy (u, v) là cạnh cầu khi nó là cạnh nét liền. Không mất tổng quát có thể coi $prev[v] = u$. Nếu $low[v] \leq num[u]$ thì bằng cách từ v xuôi theo các cạnh nét liền và ngược lên bằng cạnh nét đứt ta có thể đi đến đỉnh u mà không cần sử dụng cạnh (u, v). Với hình dung này có thể thấy điều kiện cần và đủ để cạnh (u, v) với $prev[v] = u$ là cạnh cầu là:

$$low[v] > num[u]$$

(Chứng minh chi tiết có thể xem trong tài liệu sách giáo khoa chuyên tin)
Nếu bỏ đi các cạnh cầu, phần đồ thị còn lại sẽ là các thành phần **song liên thông** - là các thành phần S mà với hai đỉnh $x, y \in S$ luôn có hai đường đi không chung cạnh từ x đến y.
Như vậy ta có thể hình dung đồ thị vô hướng như là tập hợp các thành phần song liên thông được nối với nhau bởi các cạnh cầu. Nếu ta coi mỗi thành phần song liên thông là một đỉnh của đồ thị ta có được một đồ thị có dạng hình cây.

Đoạn code dưới đây cho phép xử lý các cạnh cầu:

```
for u:=1 to n do
for v ∈ ke(u) do if (prev[v]=u) and (low[v]>num[u]) then
Xử lý (u,v) như là một cạnh cầu:
```

2) Đỉnh khớp

Một đỉnh u được gọi là một đỉnh khớp (hay còn gọi là đỉnh bản lề) nếu như bỏ u và các cạnh liên thuộc với u khỏi đồ thị thì số thành phần liên thông của đồ thị tăng lên.
Ta có thể mô tả các đỉnh khớp bằng mảng **hinge[1..maxn]** với $hinge[u] = k$ có nghĩa là khi bỏ đỉnh u và các đỉnh liên thuộc với nó thì số thành phần liên thông của đồ thị tăng lên k. Tất nhiên nếu lập được mảng trên thì đỉnh u là đỉnh khớp nếu như $hinge[u] > 0$ (lúc đó $hinge[u]$ là bậc của khớp)
Để tính mảng hinge[...] ta xét hai trường hợp:
+) $prev[u] = 0$. Lúc này u là đỉnh gốc của cây DFS. Tất nhiên dễ thấy rằng nếu cây này có hai nhánh xuất phát từ gốc thì khi bỏ gốc đi hai nhánh này sẽ rời nhau - u là đỉnh khớp và số nhánh sẽ là số thành phần liên thông mới

+) $prev[u] = w \neq 0$. Gọi $v_1, v_2, \ldots, v_k$ là các "con" của u (tức $prev[v_i] = u$) khi đó nếu như ta có $low[v_i] \geq num[u]$ thì nếu bỏ đi u khỏi đồ thị v_i không bao giờ đến được w - đỉnh u là đỉnh khớp và số lượng $low[v_i] \geq num[u]$ chính là bậc của khớp

Đoạn giả lệnh dưới đây để tìm mảng hinge:

```
for u:=1 to n do
begin
    hinge[u]:=0;
    if prev[u]=0 then
    begin
        for v ∈ ke(u) do if prev[v]=u then inc(hinge[u]);
        dec(hinge[u]);
    end else
    begin
        for v ∈ ke(u) do
        if (prev[v=u) and (low[v]>=num[u]) inc(hinge[u]);
    end;
end;
```

3) Qui hoạch động bằng cách sử dụng topo DFS

Chú ý rằng mảng **tp[1..maxn]** cho ta một thứ tự topo tren cây DFS (nếu coi mỗi cạnh là định hướng từ đỉnh prev[u] tới u) và *tất cả các đỉnh nằm trong cây con gốc u tạo thành một đoạn liên tiếp trên mảng tp*. Điều này cho phép chúng ta có thể thực hiện các giải thuật qui hoạch động trên thứ tự này. Các giải thuật này được chia thành các nhóm sau:

a) Cập nhật thông tin đỉnh cha từ các đỉnh con (qui hoạch ngược):

```
Khởi tạo thông tin tất cả các đỉnh
for i:=n downto 1 do
begin
    u:=tp[i];
    for v ∈ ke(u) if prev[v]=u then
    Cập nhật thông tin của u qua v
 end;
```

b) Cập nhật thông tin của đỉnh con từ đỉnh cha (qui hoạch động xuôi)

```
for i:=1 to n do
begin
    u:=tp[i];
    for v ∈ ke(u) if prev[v]=u then
    Cập nhật thông tin đỉnh v từ đỉnh u
end;
```

c) Xử lý cây con như là một đoạn liên tục

Dựa trên tính chất các đỉnh của cây con gốc u bất kỳ là một đoạn liên tục trên mảng tp ta có thể đưa bài toán thay đổi thông tin trên toàn bộ cây con thành bài toán cập nhật dữ liệu trên một đoạn liên tục. Từ đó có thể sử dụng các cấu trúc dữ liệu cho đoạn liên tục như segment tree, binary indexed tree,...

Một điều thú vị là trong quá trình code chúng ta có thể tách thành hai phần độc lập: code DFS để tìm các mảng cần thiết là dựa trên các mảng này phần thứ hai xử lý tiếp độc lập với quá trình DFS. Như vậy chương trình nhìn chung sẽ dễ debug hơn.

Cuối cùng, thuật toán Tarjan cho phép chúng ta tìm các thành phần liên thông mạnh cũng như các thành phần song liên thông. Hai kết quả dưới đây là đáng quan tâm:

- Nếu coi mỗi thành phần liên thông mạnh là một đỉnh ta có được một đồ thị mới - đồ thị có hướng không chu trình (DAG) và **mảng tp chính là một sắp xếp topo trên DAG này**
- Nếu coi mỗi thành phần song liên thông trên đồ thị vô hướng là một đỉnh ta có một cây vô hướng và **số hiệu các thành phần song liên thông theo thứ tự giảm dần chính là một thứ tự duyệt DFS trên cây nói trên**