

# Thuật toán đồ thị cơ bản

TÀI LIỆU

SCIENCE AND TECHNOLOGY

Thích 1

Chia sẻ 1

Tweet

G+

0

## Biểu diễn đồ thị

### Các phương pháp biểu diễn đồ thị

Có hai chuẩn dùng để biểu diễn đồ thị  $G = (V, E)$ :

Sử dụng tập hợp các danh sách kề:

Danh sách kề biểu diễn các đỉnh kề nhau, và thường hay được sử dụng bởi vì nó biểu diễn được một cách tối ưu đồ thị thưa (đồ thị thưa là đồ thị có số cạnh  $|E|$  ít hơn bình phương số các đỉnh  $|V|^2$ ). Hầu hết các thuật toán đồ thị được trình bày trong các phần sau sẽ sử dụng danh sách kề để biểu diễn đồ thị.

Sử dụng ma trận kề:

Nên sử dụng cách biểu diễn này đối với đồ thị dày, có nghĩa là khi  $|E|$  gần bằng  $|V|^2$ , hoặc khi cần biết giữa hai đỉnh nào đó có cạnh nối hay không. Một số thuật toán tìm đường đi ngắn nhất sử dụng cách biểu diễn đồ thị đầu vào  $G$  qua ma trận kề.

### Danh sách kề

Danh sách kề (là danh sách các đỉnh kề của đồ thị  $G = (V, E)$ ) được mô tả bởi mảng  $Adj$  của  $|V|$  danh sách, mỗi một danh sách tương ứng với mỗi một đỉnh thuộc  $V$ . Với mỗi đỉnh  $u \in V$ , danh sách kề  $Adj[u]$  chứa (hay còn nói là trữ tới) tất cả các đỉnh  $v$  mà từ  $u$  có cạnh nối tới  $v$  (có nghĩa là  $(u, v) \in E$ ). Điều này có nghĩa là  $Adj[u]$  chứa tất cả các đỉnh kề với  $u$  trên đồ thị  $G$ . Các đỉnh trong mỗi một danh sách kề thông thường được lưu theo một trật tự tùy ý.

Nếu  $G$  là đồ thị có hướng, tổng chiều dài của tất cả các danh sách kề là  $|E|$ , do có cạnh nối  $(u, v)$  chỉ khi đỉnh  $v$  xuất hiện trong  $Adj[u]$ . Nếu  $G$  là một đồ thị vô hướng, tổng độ dài của tất cả các danh sách kề là  $2|E|$ , do tồn tại cạnh vô hướng  $(u, v)$  chỉ khi đỉnh  $u$  xuất hiện trong danh sách các đỉnh kề của  $v$  và đỉnh  $v$  xuất hiện trong danh sách các đỉnh kề của  $u$ . Dù đồ thị có hướng hay vô hướng, việc biểu diễn qua danh sách kề chiếm một vùng bộ nhớ có kích thước là  $O(\max(V, E)) = O(V + E)$ .

Các danh sách các đỉnh kề có thể dễ dàng được sử dụng để biểu diễn các đồ thị có trọng số. Đó là đồ thị mà mỗi cạnh đều có một trọng số riêng, được tính bằng hàm trọng số  $w: E \rightarrow \mathbb{R}$ .

Cho  $G = (V, E)$  là một đồ thị trọng số với hàm trọng số  $w$ . Trọng số  $w(u, v)$  của cạnh  $(u, v) \in E$  được chứa cùng với đỉnh  $v$  trong danh sách các đỉnh kề với đỉnh  $u$ .

Biểu diễn danh sách đỉnh kề khá mạnh, xét từ khía cạnh nó có thể được biến đổi để hỗ trợ nhiều đồ thị đa dạng khác.

Một nhược điểm của cách biểu diễn đồ thị qua danh sách các đỉnh kề là để xác nhận cạnh  $(u, v)$  đã cho có tồn tại trên đồ thị  $G(V, E)$  không, không có một cách nào nhanh hơn là tiến hành tìm kiếm đỉnh  $v$  trong danh sách đỉnh  $Adj[u]$ . Có thể khắc phục nhược điểm này bằng cách chuyển sang biểu diễn đồ thị qua ma trận kề, tuy nhiên, việc này sẽ đòi hỏi phải sử dụng nhiều bộ nhớ hơn.

### Ma trận kề

Với cách biểu diễn đồ thị  $G = (V, E)$  qua ma trận kề, quy ước rằng các đỉnh được đánh từ 1, 2, ..., cho đến  $|V|$ . Biểu diễn qua ma trận kề này được mô tả qua ma trận  $A = (a_{ij})$  có kích thước  $|V| \times |V|$  như sau:

$a_{ij} = 1$  nếu  $(i, j) \in E$

$= 0$  trong trường hợp khác

Hình 23.1(c) và hình 23.2(c) là các ma trận kề của đồ thị vô hướng và có hướng cho

trong hình 23.1(a) và 23.2(a) tương ứng. Ma trận kề của đồ thị yêu cầu  $P(V^2)$  bộ nhớ, không phụ thuộc vào số cạnh trong đồ thị.

Nếu để ý quan sát tính đối xứng qua đường chéo chính của ma trận kề trong hình 23.1(c). Định nghĩa hoán vị của ma trận  $A = (a_{ij})$  là ma trận  $AT = (a_{ij}^T)$  với  $a_{ij}^T = a_{ji}$ .

Do với đồ thị vô hướng  $(u, v)$  và  $(v, u)$  đều biểu diễn cùng một cạnh, nên ma trận kề  $A$  của một đồ thị vô hướng chính là hoán vị của nó:  $A = AT$ . Trong một vài ứng dụng, chỉ cần lưu toàn bộ dữ liệu trên đường chéo của ma trận (tạo thành hình tam giác trên), do đó bộ nhớ được giảm một nửa.

Giống như cách biểu diễn qua danh sách các đỉnh kề, cách biểu diễn qua ma trận kề có thể được sử dụng cho đồ thị có trọng số. Ví dụ, nếu  $G = (V, E)$  là một đồ thị trọng số với hàm  $w$ , trọng số  $w(u, v)$  của cạnh  $(u, v)$  được lưu trữ tương ứng với hàng  $v$  và cột  $u$  của ma trận kề. Nếu cạnh không tồn tại, một giá trị rỗng (NIL) sẽ được lưu trữ tương ứng trên ma trận. Nhiều bài toán thông thường sử dụng giá trị 0 hoặc  $\infty$ .

Mặc dù cách biểu diễn đồ thị danh sách các đỉnh kề gần như có hiệu quả ít nhất là bằng gần cách biểu diễn ma trận kề, nhưng vì tính đơn giản của cách biểu diễn ma trận kề nên cách biểu diễn này hay được sử dụng hơn đối với đồ thị khá nhỏ. Hơn thế nữa, nếu đồ thị không trọng số, lợi ích của việc sử dụng ma trận kề lại càng rõ ràng hơn: thay vì sử dụng một word nhớ cho mỗi một phần tử ma trận, ma trận kề chỉ cần sử dụng đến một bit tương ứng với một phần tử của ma trận.

## Tìm kiếm theo chiều rộng

### Khái niệm

Duyệt theo chiều rộng (Breadth – first search, BFS) là một trong những thuật toán đơn giản nhất được sử dụng trong việc duyệt và tìm kiếm trên đồ thị. Trên cơ sở của thuật toán này mà nhiều các thuật toán đồ thị quan trọng khác đã ra đời, như là:

Thuật toán Dijkstra giải bài toán đường đi ngắn nhất

Thuật toán Prim giải bài toán cây khung nhỏ nhất.

Giải sử cho đồ thị  $G=(V,E)$  vô hướng hoặc có hướng và một đỉnh xuất phát  $s$  bất kỳ. Thuật toán duyệt theo chiều sâu tiến hành:

Duyệt các cạnh của đồ thị  $G$  để tìm ra từ đỉnh xuất phát  $s$  có thể đến được các đỉnh nào. Các đỉnh này còn được gọi là các đỉnh được “thăm”.

Tính toán khoảng cách (là số ít nhất các cạnh) từ  $s$  tới tất cả các đỉnh có thể đến được từ  $s$ .

Xây dựng cây (breath-first tree) có gốc là  $s$  và chứa tất cả các đỉnh có thể đến được từ  $s$ . Với một đỉnh  $v$  được thăm bất kỳ, tồn tại một đường đi trên cây này từ gốc  $s$  tới nút tương ứng với đỉnh  $v$  đó. Đường đi này chính là đường đi ngắn nhất từ  $s$  tới  $v$  trong đồ thị  $G$ , có nghĩa là số các cạnh của đường này là ít nhất.

## Hoạt động của giải thuật

### Nguyên tắc tô màu

Sở dĩ thuật toán BFS có tên gọi như vậy là do tại mỗi một bước của thuật toán, nó mở rộng biên giới giữa các đỉnh được thăm và chưa được thăm theo một quy tắc nhất định: thuật toán lần lượt thăm các đỉnh có khoảng cách từ đỉnh xuất phát là  $k$  (có nghĩa là số cạnh từ  $s$  tới các đỉnh này là  $k$ ) trước khi thăm bất cứ một đỉnh nào có khoảng cách từ đỉnh  $s$  tới nó là  $k+1$ .

Trong quá trình hoạt động, thuật toán tiến hành bôi màu các đỉnh với một trong 3 màu - màu trắng, xám hoặc đen theo nguyên tắc sau:

Tại bước khởi động tất cả các đỉnh của đồ thị đều có màu trắng.

Tất cả các đỉnh đã được thăm đều có màu xám hoặc đen.

Thuật toán BFS phân biệt giữa các đỉnh có màu xám và màu đen để đảm bảo tính chất duyệt theo chiều rộng của mình.

Khi đỉnh  $v$  được thăm lần đầu tiên, nó được bôi màu xám. Các đỉnh này biểu hiện biên giới giữa các đỉnh đã được thăm và chưa được thăm. Sau đó thuật toán tiến hành duyệt các đỉnh kề với đỉnh  $v$ .

Đỉnh  $v$  được bôi màu đen chỉ khi duyệt xong (thăm) tất cả các đỉnh kề với  $v$ .

Như vậy là nếu có cạnh  $(u,v) \in E$  và đỉnh  $u$  có màu đen, đỉnh  $v$  có thể là xám hoặc đen.

Điều này có nghĩa là tất cả các đỉnh kề với đỉnh có màu đen đều chưa được thăm.

### Cây Breadth–First Tree

Thuật toán BFS cho phép xây dựng cây  $T$  (breadth – first tree) như sau:

Ban đầu cây này chỉ có mỗi một gốc là đỉnh  $s$ .

Trong quá trình duyệt, giả sử thuật toán đã duyệt đến đỉnh chưa được thăm  $u$ .

Khi đó  $u$  trở thành đỉnh đã được thăm và thuật toán sẽ tiến hành duyệt danh sách các đỉnh kề với đỉnh  $u$ . Nếu gặp phải đỉnh  $v$  kề với  $u$ , mà đỉnh  $v$  lại có màu trắng (chưa được thăm lần nào cả), đỉnh  $v$  và cạnh  $(u,v)$  được bổ sung vào trong cây. Điều này có nghĩa là nút tương ứng với đỉnh  $u$  là nút cha của nút tương ứng với đỉnh  $v$ , còn đỉnh  $u$  được gọi là đỉnh trước của đỉnh  $v$  trong thứ tự duyệt theo chiều rộng.

Do các đỉnh được thăm nhiều nhất là một lần, nên nút trong cây  $T$  tương ứng với đỉnh này chỉ có một cha.

### Mô tả thuật toán

Thuật toán BFS được mô tả như sau:

#### Đầu vào:

Cho đồ thị vô hướng hoặc có hướng  $G=(V,E)$  được biểu diễn bằng danh sách kề.

Màu của đỉnh  $u \in V$  được lưu trong mảng  $color[u]$ .

Đỉnh trước của đỉnh  $u$  được lưu trong mảng  $\pi[u]$ . Nếu đỉnh  $u$  không có đỉnh trước, khi đó  $\pi[u]=NIL$ .

Khoảng cách từ đỉnh xuất phát  $s$  tới đỉnh  $u$  được lưu trong mảng  $d[u]$ .

Thuật toán sử dụng hàng đợi  $Q$  (theo nguyên tắc FIFO – vào trước ra trước) để quản lý tập các đỉnh có màu xám.

#### Mô phỏng thuật toán BFS bằng ngôn ngữ giả Pascal:

BFS ( $G,s$ )

1 for each vertex  $u \leftarrow V[G] - \{s\}$

2 do  $color[u] \leftarrow WHITE$

3  $d[u] \leftarrow \infty$

4  $\pi[u] \leftarrow NIL$

5  $colors[s] \leftarrow GRAY$

6  $d[s] \leftarrow 0$

7  $\pi[s] \leftarrow NIL$

8  $Q \leftarrow \{s\}$

9 while  $Q \neq \emptyset$

10 do  $u \leftarrow head[Q]$

11 for each  $v \leftarrow Adj[u]$

12 do if  $color[v] = WHITE$

13 then  $color[v] \leftarrow GRAY$

14  $d[v] \leftarrow d[u] + 1$

15  $\pi[v] \leftarrow u$

16 ENQUEUE(Q,v)

17 DEQUEUE(Q)

18 color[u]<-BLACK

#### Phân tích đoạn mã:

Bước 1 (dòng 1 cho đến dòng 4):

Thực hiện vòng lặp for để khởi tạo các giá trị ban đầu cho tất cả các đỉnh khác với đỉnh xuất phát s của đồ thị (each vertex  $u \leftarrow V[G] - \{s\}$ ):

Gán màu trắng cho mọi đỉnh của đồ thị G trừ đỉnh s (color [u] <- white)

Do tại bước đầu chưa tiến hành duyệt đồ thị, nên gán khoảng cách từ đỉnh s tới các đỉnh còn lại bằng vô cùng ( $d[u] \leftarrow \infty$ )

$pi[u] \leftarrow NIL$ , có nghĩa là chưa xác định được đỉnh đứng trước đỉnh u trong thứ tự duyệt theo chiều rộng.

Bước 2 (dòng 5 cho đến dòng 7):

Khởi tạo cho đỉnh xuất phát s:

Gán màu xám cho s color[s] <- gray

Gán  $d[u] \leftarrow 0$  (khoảng cách từ s tới chính nó là 0)

Gán  $pi[s] \leftarrow nil$

Đẩy đỉnh s vào hàng đợi Q ( s trở thành phần tử đầu của Q)

Bước 3 (dòng 9 cho đến dòng 18):

Đây là bước chính của thuật toán. Nó được lặp đi lặp lại khi trên đồ thị vẫn còn các đỉnh có màu xám, có nghĩa là khi còn có các đỉnh mà danh sách kề của chúng vẫn chưa được duyệt:

1) Kiểm tra hàng đợi Q có rỗng không. Nếu rỗng, kết thúc thuật toán. Ngược lại, chuyển sang bước 2.

2) Lấy phần tử u đầu từ Q.

3) Duyệt lần lượt danh sách các đỉnh kề của đỉnh u.

4) Với mỗi phần tử v của danh sách kề này kiểm tra xem v đã được thăm chưa (đỉnh v chưa được thăm khi  $color[v] \leftarrow WHITE$ ). Nếu v chưa được thăm thì:

Tiến hành thăm đỉnh này bằng cách:

- gán  $color[v] \leftarrow GRAY$

- gán giá trị mới cho khoảng cách từ đỉnh xuất phát s tới v:  $d[v] \leftarrow d[u]+1$

- ghi nhớ lại đỉnh đứng trước đỉnh v:  $pi[v] \leftarrow u$

Đẩy đỉnh vào cuối của hàng đợi Q (dòng 16).

5) Lặp lại bước 4 với phần tử tiếp theo trong danh sách các đỉnh kề với đỉnh u cho đến khi duyệt hết danh sách này.

6) Lấy phần tử u ra khỏi hàng đợi Q.

7) Gán màu đen cho u:  $color[u] \leftarrow BLACK$

Chú ý:

Thuật toán BFS trên sẽ không duyệt hết tất cả các đỉnh của đồ thị G nếu đồ thị này gồm nhiều thành phần liên thông khác nhau.

Trong trường hợp này thuật toán BFS phải được điều chỉnh lại như sau:

Để duyệt được hết các thành phần liên thông của đồ thị G, thuật toán được điều chỉnh lại như sau:

Thêm một vòng lặp for ở ngoài cùng để duyệt mọi đỉnh của G. Đỉnh nào chưa được thăm, tiến hành đẩy đỉnh đó vào hàng đợi Q và tiến hành một số động tác duyệt danh sách kề và gán giá trị tương ứng như đã được đề cập trong thuật toán đầu.

Như vậy, xuất phát từ đỉnh s nếu đã duyệt xong tất cả các đỉnh thuộc cùng một vùng liên thông mà thuật toán vẫn còn bỏ sót các đỉnh thuộc các thành phần liên thông khác, thuật toán kiểm tra xem có còn các đỉnh bị bỏ sót không (khác với thuật toán đầu), nếu còn, nó sẽ tiến hành duyệt tiếp các đỉnh này.

#### Đánh giá độ phức tạp tính toán của thuật toán BFS

- Sau khi được khởi tạo, các đỉnh đều được gán màu trắng, nên mỗi một đỉnh sẽ được đưa vào trong hàng đợi Q nhiều nhất là một lần, và hiển nhiên là cũng được đưa ra khỏi hàng đợi nhiều nhất là một lần. Thao tác đẩy vào và lấy ra khỏi hàng đợi Q sẽ mất thời gian là  $O(1)$ , vì vậy, thời gian tổng cộng dành cho các phép toán với hàng đợi là  $O(V)$ .

- Do danh sách kề của mỗi một đỉnh được duyệt chỉ khi đỉnh này được đưa vào trong hàng đợi, nên danh sách kề của mỗi một đỉnh cũng được duyệt nhiều nhất là một lần. Chiều dài của tất cả các danh sách kề là  $P(E)$ . Do vậy, thời gian dành cho việc duyệt toàn bộ các danh sách kề là  $O(E)$ .

Vậy, độ phức tạp tính toán của thuật toán BFS là  $O(V+E)$ . Từ đó cũng suy ra thời gian tính toán của BFS tỷ lệ tuyến tính với kích thước của danh sách kề của đồ thị G.

## Tìm kiếm theo chiều sâu

### Khái niệm

Thuật toán tìm kiếm theo chiều sâu đúng như cái tên của nó nghĩa là tìm kiếm sâu dần trên đồ thị chừng nào còn có thể. ý tưởng của thuật toán có thể được trình bày như sau:

Ta sẽ bắt đầu tìm kiếm từ một đỉnh  $v_0$  nào đó của đồ thị. Sau đó chọn u là một đỉnh kề với  $v_0$  và lặp lại quá trình đối với u. ở bước tổng quát giả sử ta đang xét đỉnh v. Trong thuật toán tìm kiếm theo chiều sâu, các cạnh đến được đỉnh v sẽ chưa được thăm cho đến khi nào v còn cạnh chưa được thăm. Khi tất cả các cạnh của v đã được thăm, thì ta nói rằng đỉnh v đã được duyệt xong và quay trở lại tìm kiếm từ đỉnh mà trước đó ta đến được đỉnh v. Quá trình này tiếp tục cho đến khi  $v = v_0$  thì kết thúc nghĩa là tất cả các cạnh của đồ thị đã được thăm.

Trong thuật toán tìm kiếm theo chiều sâu, với  $u$  là một đỉnh đã được thăm, thì mỗi khi một đỉnh  $v$  thuộc danh sách kề của  $u$  được thăm, thuật toán sẽ đánh dấu bằng cách đặt trước  $v$  một trường  $\pi v$  có giá trị là  $u$ . Khác với thuật toán tìm kiếm theo chiều rộng toàn bộ đồ thị con trước đó (predecessor subgraph – là đồ thị tạo ra khi thăm các đỉnh trước đó trong quá trình tìm kiếm) xác định một cây tìm kiếm duy nhất, trong thuật toán tìm kiếm theo chiều sâu, đồ thị con trước đó có thể xác định nhiều cây tìm kiếm khác nhau bởi thuật toán có thể được lặp lại từ nhiều đỉnh khác nhau. Vì vậy đồ thị con trước đó trong thuật toán tìm kiếm theo chiều sâu được định nghĩa hơi khác một chút so với trong thuật toán tìm kiếm theo chiều rộng. Ta định nghĩa như sau:

$G\pi = (V, E\pi)$  trong đó  $E\pi = \{ (\pi[v], v) : v \in V \text{ và } \pi[v] \neq \text{NIL} \}$

Đồ thị con trước đó trong thuật toán tìm kiếm theo chiều sâu sẽ xác định một rừng cây tìm kiếm theo chiều sâu ( a depth-first forest ) là tập hợp của các cây tìm kiếm theo chiều sâu ( depth- first trees ). Các cạnh thuộc  $E\pi$  gọi là các cạnh của cây (tree edges).

Trong thuật toán tìm kiếm theo chiều sâu các đỉnh của đồ thị được tô màu để mô tả trạng thái của nó tại mỗi thời điểm. Các đỉnh ban đầu chưa được thăm sẽ được khởi tạo là màu trắng., khi được thăm sẽ tô màu xám và tô màu đen khi đã duyệt xong. Giải thuật tô màu trên sẽ đảm bảo chính xác mỗi đỉnh chỉ được duyệt một lần, vì vậy mà các cây tìm kiếm theo chiều sâu phân biệt được với nhau.

Bên cạnh việc tạo ra một rừng cây tìm kiếm theo chiều sâu, thuật toán tìm kiếm theo chiều sâu còn gán cho mỗi đỉnh một tem thời gian (timestamps). Mỗi đỉnh  $v$  sẽ có 2 tem thời gian: tem thứ nhất  $d[v]$  sẽ được gán khi đỉnh  $v$  được thăm lần đầu tiên ( được tô màu xám), tem thứ hai  $f[v]$  sẽ được gán khi các đỉnh trong danh sách kề của  $v$  đã được duyệt ( $v$  được tô màu đen). Các tem thời gian này được sử dụng nhiều trong các thuật toán đồ thị và cũng giúp ích rất nhiều cho việc mô tả quá trình thực hiện các bước trong thuật toán tìm kiếm theo chiều sâu.

Thủ tục DFS dưới đây sẽ ghi lại thời điểm thăm đỉnh  $u$  lần đầu tiên trong biến  $d[u]$  và thời điểm đỉnh  $u$  đã duyệt xong trong biến  $f[u]$ . Giá trị của 2 tem thời gian  $d[u]$ ,  $f[u]$  là các số nguyên nằm trong khoảng từ 1 đến  $2 \cdot |V|$ . Với mỗi đỉnh  $u$ :  $d[u] < f[u]$ , đỉnh  $u$  được tô màu trắng trước thời điểm  $d[u]$ , màu xám trong khoảng thời gian giữa  $d[u]$  và  $f[u]$ , và màu đen sau thời điểm  $f[u]$ .

Thủ tục mô phỏng thuật toán tìm kiếm theo chiều sâu, trong đó đồ thị  $G$  ban đầu có thể vô hướng hoặc có hướng. Biến time là biến chung được chúng ta sử dụng cho việc gán tem thời gian.

DFS( $G$ )

1 for  $u \in V[G]$  do

2 do  $\text{color}[u] \leftarrow \text{WHITE}$

3  $[u] \leftarrow \text{NIL}$

4  $\text{time} \leftarrow 0$

5 for  $u \in V[G]$  do

6 if  $\text{color}[u] = \text{WHITE}$

7 then DFS-Visit( $u$ )

**DFS - Visit( $u$ )**

1  $\text{color}[u] \leftarrow \text{GRAY}$  Trong khi đỉnh  $u$  chưa được thăm.

2  $d[u] \leftarrow \text{time} \leftarrow \text{time} + 1$

3 for  $v \in \text{Adj}[u]$  thăm cạnh  $(u,v)$

4 do if  $\text{color}[v] = \text{WHITE}$

5 then  $[v] \leftarrow u$

6 DFS-Visit( $v$ )

7  $\text{color}[u] \leftarrow \text{BLACK}$  Tô màu đen cho đỉnh  $u$  khi đã duyệt xong

8  $f[u] \leftarrow \text{time} \leftarrow \text{time} + 1$

Thủ tục DFS làm việc như sau:

Dòng 1-3 sẽ tô màu tất cả các đỉnh của đồ thị là màu trắng và khởi tạo trường  $\pi$  của mỗi đỉnh về giá trị NIL.

Dòng 4 sẽ khởi tạo lại biến đếm thời gian chung.

Dòng 5-7 sẽ kiểm tra từng đỉnh thuộc tập  $V$ , khi một đỉnh màu trắng được tìm thấy, thực hiện thăm đỉnh bằng thủ tục DFS-Visit. Mỗi khi thủ tục DFS- Visit( $u$ ) được gọi tại dòng 7, đỉnh  $u$  sẽ trở thành gốc của một cây tìm kiếm mới trong rừng cây tìm kiếm theo chiều sâu.

Khi thủ tục DFS kết thúc, mỗi đỉnh  $u$  sẽ được gán hai tem thời gian thời gian là  $d[u]$  và  $f[u]$ .

Tại mỗi lần gọi thủ tục DFS-Visit( $u$ ):

Đỉnh  $u$  ban đầu đang được tô màu trắng.

Khi thực hiện, dòng 1 sẽ tô màu xám cho đỉnh  $u$

Dòng 2 ghi lại thời điểm đỉnh  $u$  bắt đầu được thăm lần đầu tiên vào biến time sau khi tăng biến lên 1.

Dòng 3-6 kiểm tra các đỉnh  $v$  kề với đỉnh  $u$  và thực hiện thăm  $v$  một cách đệ quy nếu nó vẫn là đỉnh trắng. Với mỗi đỉnh  $v$  được xét tại dòng 3, ta nói rằng cạnh  $(u,v)$  đã được thăm trong thuật toán tìm kiếm theo chiều sâu.

Khi tất cả các cạnh đi từ  $u$  đã được thăm, dòng 7-8 sẽ tô màu đen cho đỉnh  $u$  và ghi lại thời điểm đỉnh  $u$  đã duyệt xong trong biến  $f[u]$ .

## Độ phức tạp tính toán

Để đánh giá được độ phức tạp tính toán của thủ tục DFS, trước hết nhận thấy rằng số phép toán cần thực hiện trong hai chu trình của thuật toán ( hai vòng for ở dòng 1-2 và 5-7) là cỡ  $\Theta(V)$ . Thủ tục DFS-Visit sẽ được gọi chính xác chỉ một lần cho mỗi đỉnh thuộc  $V$  và mỗi lần thực hiện không quá  $|\text{Adj}[v]|$  phép toán. Trong đó :

$$\sum_{v \in V} |Adj[v]| = \theta(E)$$

Nghĩa là tổng số phép toán cần thực hiện tại các dòng 2-5 của thủ tục DFS-Visit là  $\theta(E)$ . Vì vậy độ phức tạp tính toán của thủ tục DFS sẽ là  $\theta(V+E)$ .

### Một số định lý của thuật toán tìm kiếm theo chiều sâu

Quá trình tìm kiếm theo chiều sâu trên đồ thị mô tả nhiều thông tin về cấu trúc của đồ thị. Một trong các thuộc tính cơ bản nhất của tìm kiếm sâu là đồ thị con trước đó  $G_{\pi}$  sẽ xác định một rừng cây tìm kiếm sâu dần, trong đó cấu trúc của các cây tìm kiếm sâu phản ánh cấu trúc của các lệnh gọi đệ quy thủ tục DFS-VISIT. Nghĩa là  $u = \pi[v]$  khi và chỉ khi thủ tục DFS-VISIT được gọi trong suốt quá trình tìm kiếm các đỉnh thuộc danh sách kề của  $u$ .

Để dễ hiểu ta xét ví dụ với đồ thị chỉ có 2 đỉnh là  $u, v$ : khi thăm  $u$  danh sách kề của  $u$  là  $v$ . Theo thủ tục DFS-VISIT thì quá trình tìm kiếm có thể hiểu như sau: Bước khởi tạo sẽ tô màu trắng cho các đỉnh và gán các đỉnh trước của nó là rỗng. Sau khi gọi, thủ tục DFS-VISIT đầu tiên sẽ tô màu cho  $u$  là xám (thủ tục bình thường thì thường có các hành động như xem xét kết thúc... nhưng thủ tục ở đây đưa ra mang tính tổng quan tiếp sau chúng ta không đề cập nữa), sau đó duyệt trên danh sách kề của  $u$  thấy  $v$  thăm tiếp và tô  $v$  màu xám. Khi xét danh sách kề của  $v$  thấy không còn đỉnh nào, tô màu đen cho đỉnh  $v$  và quay trở lại thăm gốc  $u$ ... Như vậy ta có thể thấy nó tạo thành một cây có 2 nút  $u, v$ , và quá trình thăm mỗi nút chính là thực hiện gọi thủ tục đệ quy DFS-VISIT.

Đối với đồ thị tổng quát nhiều đỉnh hơn thì cũng thực hiện tương tự.

Một thuộc tính quan trọng khác của tìm kiếm sâu trên đồ thị là các lần thăm và kết thúc thăm đỉnh đều có cấu trúc ngoặc đơn (parenthesis structure). Nghĩa là nếu chúng ta biểu diễn quá trình bắt đầu thăm một đỉnh  $u$  (chẳng hạn có thể dùng  $stack, \dots$ ) bằng một dấu ngoặc trái trước  $u$ : " $(u$ ", quá trình kết thúc thăm  $u$  bằng một dấu ngoặc phải sau  $u$ : " $)$ ", thì toàn bộ quá trình thăm và kết thúc thăm sẽ tạo ra một biểu thức có các dấu ngoặc đơn xếp lồng nhau và đối xứng hay còn gọi là có biểu thức cấu trúc ngoặc đơn.

Định lý cấu trúc ngoặc đơn được đưa ra như sau:

#### Định lý 1 (Định lý dấu ngoặc đơn)

Khi tìm kiếm theo chiều sâu trên một đồ thị có hướng hay vô hướng  $G=(V,E)$ , với 2 đỉnh  $u$  và  $v$  bất kỳ, thì sẽ xảy ra một trong 3 trường hợp sau: Các khoảng  $[d[u], f[u]]$  và  $[d[v], f[v]]$  hoàn toàn rời nhau.

Khoảng  $[d[u], f[u]]$  hoàn toàn nằm trong khoảng  $[d[v], f[v]]$ , và  $u$  là con của  $v$  trong cây tìm kiếm sâu

Khoảng  $[d[v], f[v]]$  hoàn toàn nằm trong khoảng  $[d[u], f[u]]$ , và  $v$  là con của  $u$  trong cây tìm kiếm sâu

#### Chứng minh:

Chúng ta bắt đầu với trường hợp  $d[u] < d[v]$ . Có 2 khả năng xảy ra:

$d[v] < f[u]$ : trường hợp này  $v$  đã được thăm trong khi  $u$  vẫn được tô màu xám. Có nghĩa là  $v$  là con của  $u$ . Thêm vào đó, do  $v$  được thăm trong lần gần nhất hơn  $u$ , khi tất cả các cạnh của nó đã được duyệt, quá trình thăm  $v$  kết thúc, đỉnh  $v$  được duyệt xong trước khi quá trình tìm kiếm tiếp tục quay trở lại duyệt đỉnh  $u$ . Như vậy trong trường hợp này  $[d[v], f[v]]$  hoàn toàn trong khoảng  $[d[u], f[u]]$ .

Khả năng thứ 2 là  $f[u] < d[v]$ . Nghĩa là khi đỉnh  $u$  đã được tô màu đen rồi nhưng đỉnh  $v$  vẫn tô màu trắng - tương ứng với trường hợp các khoảng  $[d[u], f[u]]$  và  $[d[v], f[v]]$  hoàn toàn rời nhau.

Chúng ta chứng minh tương tự với trường hợp  $d[v] < d[u]$  vì  $u, v$  cùng vai trò. Thay đổi vai trò  $u, v$  ta có điều phải chứng minh.

#### Hệ quả 2

Đỉnh  $v$  được gọi là con thực sự của  $u$  trong rừng tìm kiếm sâu của một đồ thị có hướng hoặc vô hướng  $G$  khi và chỉ khi  $d[u] < d[v] < f[v] < f[u]$ .

**Chứng minh:** áp dụng định lý 1 ta có điều phải chứng minh

#### Định lý 3 (Định lý đường đi trắng)

Trong rừng tìm kiếm sâu của một đồ thị có hướng hoặc vô hướng  $G=(V,E)$ , đỉnh  $v$  là một đỉnh con của đỉnh  $u$  khi và chỉ khi tại thời điểm  $d[u]$  nghĩa là tại thời điểm khi bắt đầu thăm đỉnh  $u$ , đỉnh  $v$  có thể tới được từ  $u$  theo một con đường gồm toàn bộ các đỉnh tô màu trắng.

#### Chứng minh:

Điều kiện cần:

Giả sử đỉnh  $v$  là một con của  $u$  và  $w$  là một đỉnh bất kì trên đường đi giữa  $u$  và  $v$  trong cây tìm kiếm sâu, có nghĩa là  $w$  là con của  $u$ . Theo hệ quả 2,  $d[u] < d[w]$  do đó tại thời điểm  $d[u]$  đỉnh  $w$  vẫn tô màu trắng.

Điều kiện đủ:

Giả sử tại thời điểm  $d[u]$ , từ đỉnh  $u$  ta có thể đến được đỉnh  $v$  dọc theo một con đường gồm các đỉnh vẫn tô màu trắng, nhưng  $v$  không phải là con của  $u$  trong cây tìm kiếm sâu. Không mất tính tổng quát, giả sử mọi đỉnh khác trên đường đi trắng đó đều là con của  $u$ . (nếu không thì cho  $v$  là đỉnh gần  $u$  nhất dọc theo đường đi trắng nhưng không là con  $u$ ). Cho  $w$  là đỉnh trước của  $v$  trên đường trắng, mà  $w$  là con của  $u$  ( $w$  và  $u$  có thể cùng là một đỉnh), theo hệ quả 2:  $f(w) \leq f(u)$ . Chú ý rằng  $v$  phải được thăm sau khi  $u$  đã được thăm, nhưng trước khi  $w$  thăm xong. Do đó  $d[u] < d[v] < f[w] \leq f[u]$ . Từ Định lý 1 ta thấy  $[d[v], f[v]]$  phải nằm trong khoảng  $[d[u], f[u]]$ . Mà theo hệ quả thì như vậy  $v$  phải là con của  $u$ .

Ta có điều cần chứng minh.

### Phân loại cạnh

Một trong các thuộc tính đáng quan tâm của tìm kiếm sâu là quá trình tìm kiếm có thể sử dụng để phân loại các cạnh của đồ thị đầu vào  $G=(V,E)$ . Việc phân loại các cạnh có thể sử dụng để thu thập thông tin về đồ thị. Ví dụ, trong phần tiếp theo chúng ta sẽ thấy một đồ thị có hướng là không có chu trình nếu quá trình tìm kiếm sâu không gồm các cạnh **back**.

Chúng ta có thể định nghĩa 4 loại cạnh trong rừng tìm kiếm sâu  $G_{\pi}$ . Khi tìm kiếm sâu trên đồ thị  $G$ .

Các cạnh **cây** là các cạnh trong rừng tìm kiếm sâu  $G_{\pi}$ . Cạnh  $(u,v)$  là một cạnh của cây nếu  $v$  được thăm lần đầu khi duyệt cạnh  $(u,v)$ .

Cạnh **Back** là những cạnh  $(u,v)$  mà đỉnh  $u$  nối tới tổ tiên  $v$  trong cây tìm kiếm sâu.

Cạnh của một đỉnh (cạnh nối một đỉnh với chính nó) cũng được coi như là cạnh Back. Cạnh **Forward** là các cạnh không phải là cạnh cây (u,v) nối một đỉnh u tới con v trong một cây tìm kiếm sâu.

Cạnh **Chéo** là tất cả các cạnh khác các loại cạnh trên. Chúng có thể là cạnh giữa các đỉnh trong cùng một cây tìm kiếm sâu khi một đỉnh không là tổ tiên của đỉnh khác, chúng cũng có thể là cạnh giữa các đỉnh trong các cây tìm kiếm sâu khác nhau.

Thuật toán DFS có thể thay đổi để giúp ta có thể phân loại các cạnh khi thực hiện tìm kiếm sâu trên đồ thị. ý tưởng chủ đạo của nó là mỗi cạnh (u,v) có thể phân lớp bằng màu của đỉnh v - tức là đỉnh tới được khi thăm cạnh lần đầu tiên - (ngoại trừ các cạnh forward và chéo không thể phân biệt được): Màu trắng chỉ một cạnh của cây

Màu xám chỉ một cạnh **back**

Màu đen chỉ một cạnh **forward** hay một cạnh chéo

Trường hợp 1 xuất phát từ đặc điểm của thuật toán. Trường hợp 2 là do khi quan sát quá trình tìm kiếm sâu ta thấy các đỉnh tô màu xám luôn tạo ra một chuỗi các đỉnh con tương ứng với stack của đỉnh đang gọi thủ tục DFS\_VISIT. Số các đỉnh xám lớn hơn 1 so với độ sâu rừng tìm kiếm sâu của đỉnh được thăm gần đây nhất. Quá trình duyệt được tiếp tục từ đỉnh xám có độ sâu lớn nhất qua một cạnh để tới đỉnh xám khác,... rồi đến đỉnh xám tổ tiên. Trường hợp thứ 3 là các khả năng còn lại, cạnh (u,v) là cạnh forward nếu  $d[u] < d[v]$  và là cạnh chéo nếu  $d[u] > d[v]$ .

#### Định lý 4

Khi tìm kiếm theo chiều sâu trên một đồ thị vô hướng G, mọi cạnh của G hoặc là cạnh cây hoặc là cạnh back.

#### Chứng minh:

Cho (u, v) là một cạnh tùy ý của G, không làm mất tính tổng quát ta có thể giả sử  $d[u] < d[v]$ . Đỉnh v phải được thăm và duyệt xong trước khi đỉnh u duyệt xong, do đó v thuộc danh sách kề của u. Nếu cạnh (u,v) được thăm trước theo hướng u tới v thì (u,v) trở thành cạnh cây. Nếu cạnh (u,v) được thăm trước theo hướng v tới u, thì sau đó (u,v) sẽ là cạnh back, khi đó u vẫn tô màu xám tại thời điểm cạnh được thăm lần đầu tiên.

Chúng ta sẽ còn thấy rất nhiều ứng dụng của các định lý trên trong các phần sau.

## Sắp xếp Topo

### Giải thuật sắp xếp Topo

Sau đây là giải thuật sắp xếp tôpô cho một đồ thị dag:

#### Topological\_sort (G)

Gọi DFS(G) để tính các thời điểm xem xét  $f[v]$  cho mỗi đỉnh v

Với mỗi điểm đã được xem xét, cho nó vào đầu của một danh sách liên kết

Trở lại danh sách liên kết các đỉnh

Hình 23.7(b) chỉ ra cách các đỉnh được sắp xếp tôpô xuất hiện theo thứ tự ngược với thời điểm kết thúc chúng.

### Độ phức tạp tính toán

Chúng ta có thể đánh giá độ phức tạp tính toán của giải thuật là  $O(V+E)$  vì: tìm kiếm theo chiều sâu mất  $O(V+E)$  và để chèn một đỉnh (trong số |V| đỉnh) vào trước danh sách liên kết mất  $O(1)$ .

Chúng ta chứng minh tính đúng đắn của giải thuật này bằng cách sử dụng một bổ đề quan trọng nêu lên đặc tính của các đồ thị không chu trình có hướng.

#### Bổ đề 5

Một đồ thị có hướng G là không có chu trình nếu và chỉ nếu thực hiện tìm kiếm theo chiều sâu trên G sẽ không có các cạnh back.

#### Chứng minh.

##### Điều kiện cần:

Giả sử rằng có một cạnh back (u,v). Khi đó đỉnh v là đỉnh trước (đỉnh tổ tiên) của đỉnh u trong tìm kiếm theo chiều sâu. Vì thế sẽ có một đường đi từ v tới u trong G, do vậy khi kết hợp cạnh (u,v) sẽ tạo thành một chu trình.

##### Điều kiện đủ:

Giả sử rằng G chứa một chu trình c. Chúng ta sẽ chỉ ra rằng khi tìm kiếm theo chiều sâu trên G sẽ có một cạnh back. Gọi v là đỉnh đầu tiên được thăm trong c, và gọi (u, v) là cạnh trước c. Tại thời điểm  $d[v]$ , có một đường đi trắng từ v tới u. Theo định lý đường đi trắng, đỉnh u trở thành đỉnh trước của đỉnh v trong tìm kiếm theo chiều sâu. Vì thế (u, v) là một cạnh back.

#### Định lý 6

Giải thuật sắp xếp tôpô TOPOLOGICAL\_SORT(G) ở trên tạo ra một sắp xếp tôpô đối với đồ thị không chu trình có hướng G.

#### Chứng minh

Giả sử rằng thủ tục DFS được thực hiện trên một đồ thị dag  $G=(V,E)$  cho trước để xác định thời điểm kết thúc thăm các đỉnh của đồ thị. Ta sẽ chỉ ra bất cứ một cặp đỉnh riêng biệt  $u, v \in V$ , nếu có một cạnh trong G từ u tới v thì  $f[v] < f[u]$ .

Xét một cạnh bất kỳ (u,v) được thăm bởi DFS(G). Khi cạnh này được thăm, đỉnh v không thể có màu xám vì khi đó v có thể là đỉnh trước u và (u,v) sẽ là một cạnh back, mâu thuẫn. Vì thế v chỉ có thể là đỉnh màu trắng hoặc là đỉnh màu đen. Nếu v là đỉnh trắng nó thành đỉnh sau của u và vì thế  $f[v] < f[u]$ . Nếu v là đỉnh đen khi đó cũng có  $f[v] < f[u]$ . Vì thế với bất kỳ cạnh (u,v) nào trong đồ thị dag, chúng ta đều có  $f[v] < f[u]$ .

Định lý được chứng minh.

### Độ phức tạp tính toán

Chúng ta có thể đánh giá độ phức tạp tính toán của giải thuật là  $O(V+E)$  vì: tìm kiếm theo chiều sâu mất  $O(V+E)$  và để chèn một đỉnh (trong số |V| đỉnh) vào trước danh sách liên kết mất  $O(1)$ .

Chúng ta chứng minh tính đúng đắn của giải thuật này bằng cách sử dụng một bổ đề quan trọng nêu lên đặc tính của các đồ thị không chu trình có hướng.

#### Bổ đề 5

Một đồ thị có hướng  $G$  là không có chu trình nếu và chỉ nếu thực hiện tìm kiếm theo chiều sâu trên  $G$  sẽ không có các cạnh back.

#### Chứng minh.

##### Điều kiện cần:

Giả sử rằng có một cạnh back  $(u, v)$ . Khi đó đỉnh  $v$  là đỉnh trước (đỉnh tổ tiên) của đỉnh  $u$  trong tìm kiếm theo chiều sâu. Vì thế sẽ có một đường đi từ  $v$  tới  $u$  trong  $G$ , do vậy khi kết hợp cạnh  $(u, v)$  sẽ tạo thành một chu trình.

##### Điều kiện đủ:

Giả sử rằng  $G$  chứa một chu trình  $c$ . Chúng ta sẽ chỉ ra rằng khi tìm kiếm theo chiều sâu trên  $G$  sẽ có một cạnh back. Gọi  $v$  là đỉnh đầu tiên được thăm trong  $c$ , và gọi  $(u, v)$  là cạnh trước  $c$ . Tại thời điểm  $d[v]$ , có một đường đi trắng từ  $v$  tới  $u$ . Theo định lý đường đi trắng, đỉnh  $u$  trở thành đỉnh trước của đỉnh  $v$  trong tìm kiếm theo chiều sâu. Vì thế  $(u, v)$  là một cạnh back.

#### Định lý 6

Giải thuật sắp xếp tô pô TOPOLOGICAL\_SORT( $G$ ) ở trên tạo ra một sắp xếp tô pô đối với đồ thị không chu trình có hướng  $G$ .

#### Chứng minh

Giả sử rằng thủ tục DFS được thực hiện trên một đồ thị  $G=(V, E)$  cho trước để xác định thời điểm kết thúc thăm các đỉnh của đồ thị. Ta sẽ chỉ ra bất cứ một cặp đỉnh riêng biệt  $u, v \in V$ , nếu có một cạnh trong  $G$  từ  $u$  tới  $v$  thì  $f[v] < f[u]$ .

Xét một cạnh bất kỳ  $(u, v)$  được thăm bởi DFS( $G$ ). Khi cạnh này được thăm, đỉnh  $v$  không thể có màu xám vì khi đó  $v$  có thể là đỉnh trước  $u$  và  $(u, v)$  sẽ là một cạnh back, mâu thuẫn. Vì thế  $v$  chỉ có thể là đỉnh màu trắng hoặc là đỉnh màu đen. Nếu  $v$  là đỉnh trắng nó thành đỉnh sau của  $u$  và vì thế  $f[v] < f[u]$ . Nếu  $v$  là đỉnh đen khi đó cũng có  $f[v] < f[u]$ . Vì thế với bất kỳ cạnh  $(u, v)$  nào trong đồ thị  $G$ , chúng ta đều có  $f[v] < f[u]$ . Định lý được chứng minh.

## Các thành phần liên thông mạnh

### Giới thiệu

Ta xét một ứng dụng kinh điển của việc tìm kiếm theo chiều sâu: phân rã một đồ thị có hướng thành các thành phần liên thông mạnh. Trong phần này sẽ trình bày cách phân rã sử dụng hai lần tìm kiếm theo chiều sâu. Có rất nhiều các thuật toán làm việc với đồ thị có hướng sử dụng phép phân rã này; cách tiếp cận này cho phép chia một bài toán thành các bài toán nhỏ, mỗi bài toán tương ứng với một thành phần liên thông mạnh. Kết hợp các giải pháp của các bài toán nhỏ theo kiến trúc liên kết giữa các thành phần liên thông mạnh; kiến trúc này có thể được biểu diễn bởi một đồ thị được gọi là đồ thị thành phần.

Một thành phần liên thông mạnh của một đồ thị có hướng  $G=(V, E)$  là tập cực đại các đỉnh  $U \subseteq V$  sao cho mỗi cặp đỉnh  $u$  và  $v$  thuộc  $U$  ta có  $uv$  và  $vu$  có nghĩa là  $u$  và  $v$  đều có thể được đi tới từ các đỉnh khác.

Thuật toán tìm kiếm các thành phần liên thông mạnh của một đồ thị  $G=(V, E)$  sử dụng đồ thị đảo của  $G$ , đồ thị này gọi là  $GT = (V, ET)$ , trong đó  $ET = \{(u, v) : (v, u) \in E\}$ .  $ET$  bao gồm các cung của đồ thị  $G$  với chiều đảo ngược. Với danh sách kề biểu diễn cho đồ thị  $G$ , thời gian để xây dựng  $GT$  là  $O(V+E)$ . Có một điều khá thú vị đó là  $G$  và  $GT$  có cùng số lượng các thành phần liên thông:  $u$  và  $v$  có thể được đi tới từ các đỉnh thuộc  $G$  nếu và chỉ nếu chúng có thể được đi tới từ các đỉnh thuộc  $GT$ . Hình 5.1(b) biểu diễn đồ thị đảo của đồ thị ở hình 5.1(a), các thành phần liên thông được tô đậm.

a) Một đồ thị có hướng  $G$ . Các thành phần liên thông mạnh của  $G$  là các vùng được tô đậm. Mỗi đỉnh được đánh nhãn là thời gian thăm và thời gian kết thúc. Các cung của cây được tô đậm.

b)  $GT$  - đồ thị đảo của  $G$ . Cây tìm kiếm chiều sâu được tính toán ở dòng 3 của STRONGLY-CONNECTED-COMPONENTS được chỉ ra với các cung của cây được tô đậm. Mỗi thành phần liên thông mạnh tương ứng với một cây tìm kiếm chiều sâu. Các đỉnh  $b, c, g$  và  $h$  được tô đậm hơn là tổ tiên của tất cả các đỉnh trong thành phần liên thông mạnh của chúng; các đỉnh này cũng là các gốc của các cây tìm kiếm chiều sâu được tạo ra do tìm kiếm theo chiều sâu trên  $GT$ .

c) Đồ thị thành phần GSCC được thu được bằng cách co mỗi thành phần của  $G$  thành một đỉnh đơn.

Thuật toán với thời gian tính tuyến tính sau tính toán các thành phần liên thông của đồ thị có hướng  $G = (V, E)$  bằng cách sử dụng 2 lần tìm kiếm theo chiều sâu, một trên  $G$  và một trên  $GT$ .

#### STRONGLY-CONNECTED-COMPONENTS( $G$ )

1. gọi DFS( $G$ ) để tính toán thời gian kết thúc  $f[u]$  cho mỗi đỉnh  $u$
2. tính  $GT$
3. gọi DFS( $GT$ ), nhưng trong vòng lặp chính của DFS, xem xét các đỉnh để giảm  $f[u]$  (như tính toán trong bước 1)
4. đưa ra các đỉnh của mỗi cây trong rừng tìm kiếm theo chiều sâu của bước 3 như một thành phần liên thông mạnh riêng rẽ

Thuật toán trông khá đơn giản và có vẻ như không liên quan gì tới các thành phần liên thông mạnh. Tuy nhiên, bí mật thiết kế và tính đúng đắn của nó sẽ được nghiên cứu trong phần tiếp theo.

## Các bổ đề và định lý cơ bản

### Bổ đề 7

Nếu 2 đỉnh cùng thuộc một thành phần liên thông mạnh thì không có cung nào giữa chúng nằm tách rời thành phần liên thông mạnh đó.

#### Chứng minh:

Giả sử  $u$  và  $v$  là 2 đỉnh thuộc cùng thành phần liên thông mạnh. Theo định nghĩa về thành phần liên thông mạnh, có các đường đi từ  $u$  tới  $v$  và từ  $v$  tới  $u$ . Gọi  $w$  là một đỉnh nằm trên đường đi  $uw$  do vậy  $w$  là đỉnh có thể được đi tới từ  $u$ . Hơn nữa, do có đường đi từ  $w$  ta biết rằng  $u$  có thể được đi tới từ  $w$  bởi đường đi  $wu$ . Do vậy,  $u$  và  $w$  là ở trong cùng một thành phần liên thông mạnh. Do  $w$  được chọn một cách tùy ý, định lý được chứng minh.

### Bổ đề 8

Trong các phép tìm kiếm theo chiều sâu, tất cả các đỉnh thuộc cùng một thành phần liên thông mạnh sẽ thuộc cùng cây tìm kiếm theo chiều sâu.

#### Chứng minh:

Với các đỉnh trong thành phần liên thông mạnh, gọi  $r$  là đỉnh đầu tiên được thăm. Do  $r$  là đỉnh đầu tiên, các đỉnh khác trong thành phần liên thông mạnh là có màu trắng tại thời điểm nó được thăm. Có các đường đi từ  $r$  tới tất cả các đỉnh còn lại trong thành phần liên thông mạnh, do các cung này không nằm tách rời thành phần liên thông

mạnh, tất cả các đỉnh thuộc chúng đều là trắng. Do vậy, theo định lý đường đi trắng, tất cả các đỉnh trong thành phần liên thông mạnh trở thành con cháu của  $r$  trong cây tìm kiếm chiều sâu

Trong phần còn lại của phần này, ký hiệu  $d[u]$  và  $f[u]$  biểu diễn cho thời gian thả ra và thời gian kết thúc được tính toán bởi lần tìm kiếm theo chiều sâu thứ nhất ở dòng 1 của thủ tục STRONGLY-CONNECTED-COMPONENTS. Tương tự ký hiệu  $uv$  biểu diễn cho sự hiện diện của một đường đi trong  $G$  chứ không phải GT.

Để chứng minh STRONGLY-CONNECTED-COMPONENTS đúng, ta giới thiệu khái niệm  $\phi(u)$  là tổ tiên của đỉnh  $u$  là một đỉnh  $w$  có thể được đi tới từ  $u$  và được kết thúc cuối cùng trong lần tìm kiếm theo chiều sâu trong dòng 1. Nói cách khác:

$\phi(u) = w$  sao cho  $uw$  và  $f[w]$  là cực đại

Chú ý rằng  $\phi(u) = u$  là có thể xảy ra do  $u$  là có thể được đi tới từ chính nó, và do vậy:  $f[u] \leq f[\phi(u)]$  (\*)

Ta cũng có thể chỉ ra rằng  $\phi(\phi(u)) = \phi(u)$  bởi lý do sau: với  $u, v \in V$ ,  $u, v$  có nghĩa là  $f[\phi(v)] \leq f[\phi(u)]$  (\*\*) do  $\{w: v \rightarrow w\} \subseteq \{w: u \rightarrow w\}$  và đỉnh tổ tiên có thời gian kết thúc lớn nhất trong tất cả các đỉnh. Do  $\phi(u)$  là có thể được đi tới từ  $u$  công thức (\*\*) có nghĩa là  $f[\phi(\phi(u))] \leq f[\phi(u)]$ . Ta cũng có  $f[\phi(u)] \leq f[\phi(\phi(u))]$  theo bất đẳng thức (\*). Do vậy  $f[\phi(\phi(u))] = f[\phi(u)]$  nên ta có  $\phi(\phi(u)) = \phi(u)$  do 2 đỉnh mà kết thúc tại cùng thời điểm thì chính là một đỉnh.

Như ta đã thấy, tất cả các thành phần liên thông mạnh có một đỉnh là tổ tiên của tất cả các đỉnh khác trong thành phần liên thông mạnh đó; đỉnh này gọi là đỉnh đại diện của thành phần liên thông mạnh đó. Trong lần tìm kiếm theo chiều sâu của  $G$ , nó là đỉnh đầu tiên của thành phần liên thông mạnh được thăm. Trong lần tìm kiếm theo chiều sâu trên GT, nó là gốc của cây tìm kiếm chiều sâu. Chúng ta sẽ chứng minh các tính chất này

Định lý đầu tiên chứng minh gọi  $\phi(u)$  là tổ tiên của  $u$

#### Định lý 9

Trong một đồ thị có hướng  $G = (V, E)$ ,  $(u)$  của mọi đỉnh  $u \in V$  trong bất kỳ phép tìm kiếm theo chiều sâu trên  $G$  là tổ tiên của  $u$ .

#### Chứng minh:

Nếu  $\phi(u) = u$ , định lý hiển nhiên đúng. Nếu  $\phi(u) \neq u$ , xem xét màu của các đỉnh tại thời điểm  $d[u]$ . Nếu  $\phi(u)$  là màu đen, thì  $f[u] < f[\phi(u)]$  mâu thuẫn với bất đẳng thức (\*). Nếu  $\phi(u)$  là xám thì đó là một tổ tiên của  $u$  và do đó định lý được chứng minh.

Ta cần chứng minh rằng  $\phi(u)$  không phải màu trắng. Có hai trường hợp, theo màu của các đỉnh trung gian trên đường đi từ  $u$  tới  $\phi(u)$ :

1. Nếu tất cả các đỉnh là màu trắng, thì  $\phi(u)$  trở thành con cháu của  $u$  theo định lý đường đi trắng. Nhưng từ đó ta có  $f[u] < f[\phi(u)]$ , mâu thuẫn với bất đẳng thức (\*)
2. Nếu có một đỉnh trung gian nào đó không phải là trắng, gọi  $t$  là đỉnh không trắng cuối cùng trên đường đi từ  $u$  tới  $\phi(u)$ . Do vậy,  $t$  phải là xám, do không có một cung nào từ một đỉnh đen tới một đỉnh trắng, và đỉnh tiếp theo  $t$  là trắng. Nhưng từ đó do có một đường đi của các đỉnh trắng từ  $t$  tới  $\phi(u)$ , và do vậy  $\phi(u)$  là một con cháu của  $t$  theo định lý đường đi trắng. Điều đó có nghĩa là  $f[t] >$

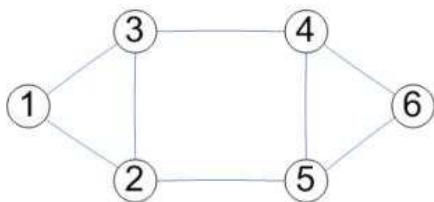
$f[u]$ , mâu thuẫn với lựa chọn của chúng ta với  $\phi(u)$ , do vậy có một đường đi từ

$u$  tới  $t$  3) và tính chất  $r = F(r)$  ta thu được  $f[\phi(w)] \geq f[\phi(r)] = \phi(r)$ , điều này mâu thuẫn với giả thiết  $f[\phi(w)] < f[r]$ .

Do vậy  $T$  sẽ chứa các đỉnh mà  $\phi(w) = r$ . Có nghĩa là  $T$  tương đương với thành phần liên thông mạnh  $C(r)$ , định lý đã được chứng minh.

### Bài tập chương

Cho đồ thị sau:

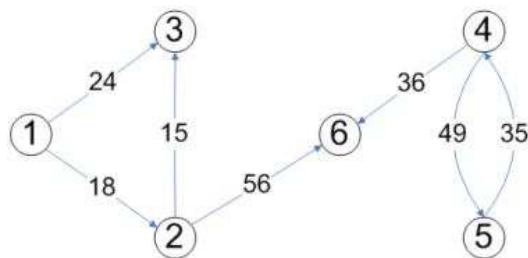


Hãy biểu diễn đồ thị bằng a). Ma trận kề

b). Danh sách kề

Dành cho đọc giả

Cho đồ thị sau



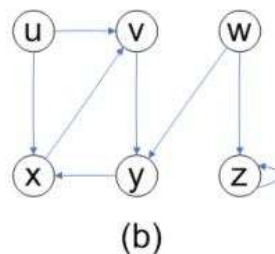
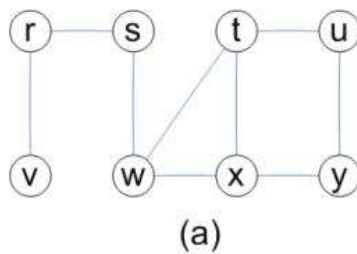


Hãy biểu diễn đồ thị bằng

c) Ma trận trọng số

d) Danh sách kề

3. Cho các đồ thị

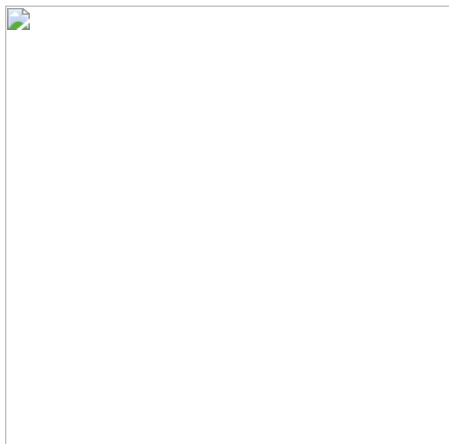


Hãy minh họa các bước của thuật toán a) Tìm kiếm theo chiều sâu

b) Tìm kiếm theo chiều rộng

Dành cho độc giả

Cho đồ thị

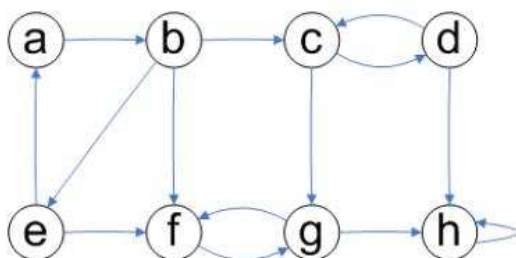


Nêu cách sắp xếp thứ tự các đỉnh do thuật toán Topological-Sort tạo ra cho hình trên.

Minh họa thuật toán.

Dành cho độc giả

Cho đồ thị



Xác định các thành phần liên thông mạnh của đồ thị

Dành cho độc giả

Thích 1 Chia sẻ 1 Tweet G+

0

0 bình luận

Sắp xếp theo 

Cũ nhất




Thêm bình luận...

Plugin bình luận của Facebook

TÀI VỆ

TÁI SỬ DỤNG(/user/reuse/m/211f7618/1)



Đại Học Phương Đông (/profile/393)

1 GIÁO TRÌNH (/PROFILE/393?TYPES=2) | 6 TÀI LIỆU (/PROFILE/393?TYPES=1)

(/profile/393)

ĐÁNH GIÁ:

0 dựa trên 0 đánh giá

Tuyển tập sử dụng module này

Bài Giảng Môn Học Phân Tích Và Thiết Kế Thuật Toán (/c/bai-giang-mon-hoc-phan-tich-va-thiet-ke-thuat-toan/d95aa558)

NỘI DUNG CÙNG TÁC GIẢ

- Độ phức tạp tính toán và tính hiệu quả của thuật toán (/m/do-phuc-tap-tinh-toan-va-tinh-hieu-qua-cua-thuat-toan/a51dc616)
- Mở đầu về thiết kế, đánh giá thuật toán và kiến thức bổ trợ (/m/mo-dau-ve-thiet-ke-danh-gia-thuat-toan-va-kien-thuc-bo-tro/f5007ea2)
- Phương pháp tham lam (/m/phuong-phap-tham-lam/f09a14a9)
- Thuật toán đồ thị cơ bản (/m/thuat-toan-do-thi-co-ban/211f7618)
- Phương pháp "chia để trị" (/m/phuong-phap-chia-de-tri/61608553)
- Quy hoạch động (/m/quy-hoach-dong/9a197ec0)
- Bài Giảng Môn Học Phân Tích Và Thiết Kế Thuật Toán (/c/bai-giang-mon-hoc-phan-tich-va-thiet-ke-thuat-toan/d95aa558)

TRƯỚC |  
TIẾP

NỘI DUNG TƯƠNG TỰ

- Ma cà rồng (/m/ma-ca-rong/64c5b529)
- Khái quát chung về ma sát (/m/khai-quat-chung-ve-ma-sat/1edac487)
- Ma trận nghịch đảo (/m/ma-tran-ngich-dao/489756e3)
- Bài thực hành về nhập xuất tệp tin trong C#.docx (/m/bai-thuc-hanh-ve-nhap-xuat-tep-tin-trong-cdocx/fb5a0e21)
- Ma sát (/m/ma-sat/9a6d3151)
- Giải hệ phương trình đại số tuyến tính (/m/giai-he-phuong-trinh-dai-so-tuyen-tinh/b5e50b2e)
- Ma sát (/m/ma-sat/b23cdd15)
- Lý thuyết ma sát và hao mòn (/m/ly-thuyet-ma-sat-va-hao-mon/9682ab2e)
- Ma (/m/ma/e921e4d1)
- Đại số ma trận ứng dụng trong giải tích mạng (/m/dai-so-ma-tran-ung-dung-trong-giai-tich-mang/12cddd94)

TRƯỚC |  
TIẾP



Thư viện Học liệu Mở Việt Nam (VOER) được tài trợ bởi Vietnam Foundation (<http://www.vnfoundation.org>) và vận hành trên nền tảng Hanoi Spring (<http://www.hanoispring.com>). Các tài liệu đều tuân thủ giấy phép Creative Commons Attribution 3.0 trừ khi ghi chú rõ ngoại lệ.

 Connect with Facebook (<https://www.facebook.com/voer.edu.vn>)



