Tìm kiếm trên đồ thị (Version 0.2)

Trần Vĩnh Đức

HUST

Ngày 7 tháng 9 năm 2017

Tài liệu tham khảo

- S. Dasgupta, C. H. Papadimitriou, and U. V. Vazirani, *Algorithms*, July 18, 2016.

- Chú ý: Nhiều hình vẽ trong tài liệu được lấy tùy tiện mà chưa xin phép.

Nội dung

Biểu diễn đồ thị

Tìm kiếm theo chiều sâu trên đồ thị vô hướng

Tìm kiếm theo chiều sâu trên đồ thị có hướng

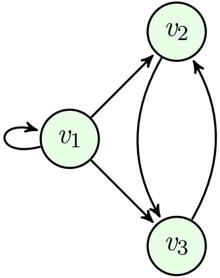
Thành phần liên thông mạnh

Biểu diễn đồ thị dùng Ma trận kề

Nếu đồ thị có $n = |V|$ đỉnh $v_1, v_2, \ldots, v_n$, thì *ma trận kề* là một mảng $n \times n$ với phần tử (i, j) của nó là

$$a_{ij} = \begin{cases} 1 & \text{nếu có cạnh từ } v_i \text{ tới } v_j \\ 0 & \text{ngược lại.} \end{cases}$$

Ví dụ



$$A = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

Dùng ma trận kề có hiệu quả?

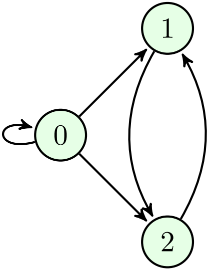
- Có thể kiểm tra có cạnh nối giữa cặp đỉnh bất kỳ chỉ cần một lần truy cập bộ nhớ.
- Tuy nhiên, không gian lưu trữ là $O(n^2)$

Biểu diễn đồ thị dùng danh sách kề

- Dùng một mảng `Adj` gồm $|V|$ danh sách.
- Với mỗi đỉnh $u \in V$, phần tử `Adj[`u`]` lưu trữ danh sách các hàng xóm của u. Có nghĩa rằng:

$$\texttt{Adj}[\texttt{u}] = \{v \in V \mid (u, v) \in E\}.$$

Ví dụ



$$\texttt{Adj}[0] = \{0, 1, 2\}$$
$$\texttt{Adj}[1] = \{2\}$$
$$\texttt{Adj}[2] = \{1\}$$

Dùng danh sách kề có hiệu quả?

- Có thể liệt kê các đỉnh kề với một đỉnh cho trước một cách hiệu quả.
- Nó cần không gian lưu trữ là $O(|V| + |E|)$. Ít hơn $O(|V|^2)$ rất nhiều khi đồ thị ít cạnh.

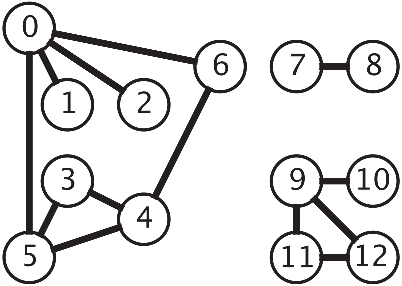
Nội dung

Biểu diễn đồ thị

Tìm kiếm theo chiều sâu trên đồ thị vô hướng

Tìm kiếm theo chiều sâu trên đồ thị có hướng

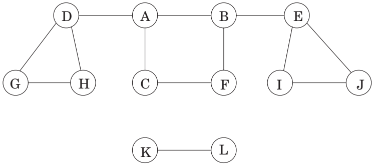
Thành phần liên thông mạnh

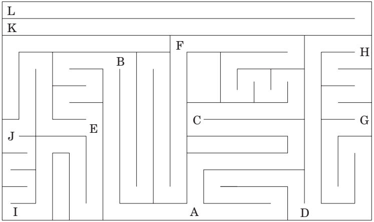


Câu hỏi

Từ một đỉnh của đồ thị ta có thể đi tới những đỉnh nào?

Tìm đường trong mê cung



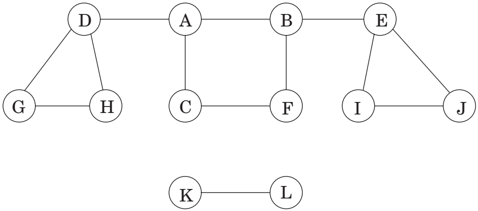


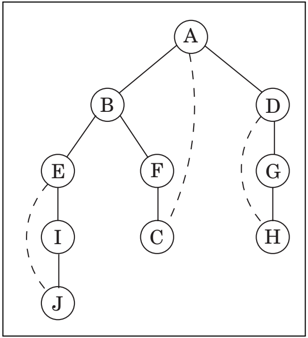
Hình: Tìm kiếm trên đồ thị cũng giống tìm đường trong mê cung

```
procedure explore(G, v)
Input: đồ thị G = (V, E); v ∈ V
Output: visited(u)=true với mọi đỉnh u có thể đến
được từ v

visited(v) = true
previsit(v)
for each edge (v, u) ∈ E:
    if not visited(u): explore(G, u)
postvisit(v)
```

Ví dụ: Kết quả chạy `explore`(A)





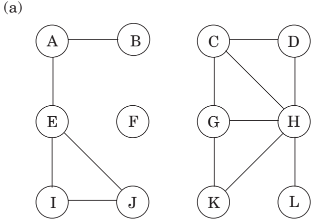
Tìm kiếm theo chiều sâu

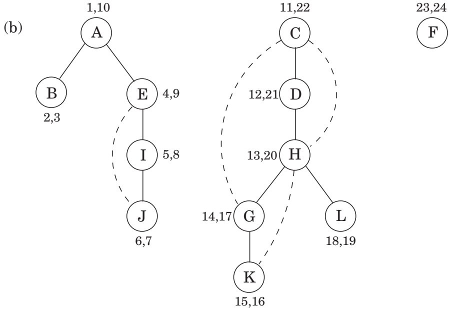
```
procedure dfs(G)

for all v ∈ V:
    visited(v) = false

for all v ∈ V :
    if not visited(v):  explore(G, v)
```

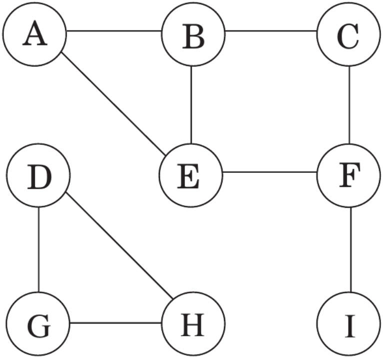
Ví dụ: Đồ thị và Rừng DFS



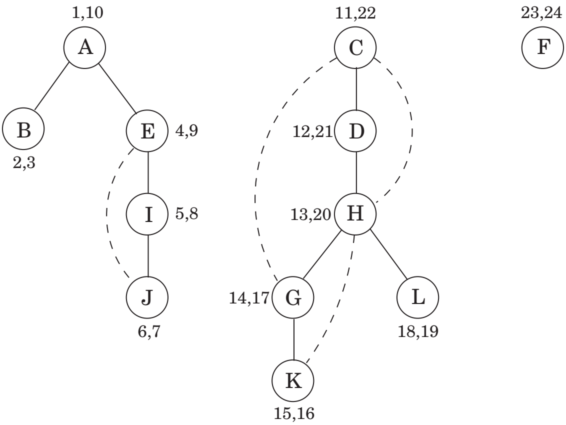


Bài tập

Xây dựng rừng DFS cho đồ thị sau. Vẽ cả những cạnh nét đứt.



Rừng DFS và số thành phần liên thông



v	ccnum[v]
A	1
B	1
C	2
D	2
E	1
F	3
G	2
H	2
I	1
J	1

Biến ccnum[v] để xác định thành phần liên thông của đỉnh v.

Tính liên thông trong đồ thị vô hướng

```
procedure dfs(G)
cc = 0
for all v ∈ V:  visited(v) = false
for all v ∈ V:
    if not visited(v):
        cc = cc + 1
        explore(G, v)

procedure explore(G, v)
visited(v) = true
previsit(v)
for each edge (v, u) ∈ E:
    if not visited(u):  explore(G, u)
postvisit(v)

procedure previsit(v)
ccnum[v] = cc
```

Bài tập

Hãy cài đặt chương trình tìm số thành phần liên thông của một đồ thị vô hướng.

previsit và postvisit

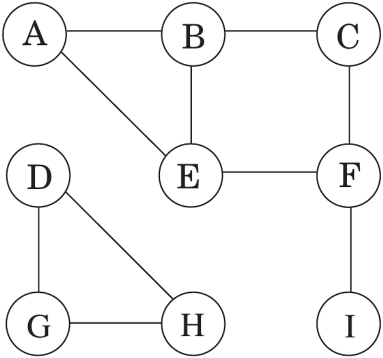
- Lưu thời gian lần đầu đến đỉnh trong mảng pre
- Lưu thời gian lần cuối rời khỏi đỉnh trong mảng post
- Để tính hai thông tin này ta dùng một bộ đếm clock, khởi tạo bằng 1, và được cập nhật như sau:

```
procedure previsit(v)
pre[v] = clock
clock = clock + 1

procedure postvisit(v)
post[v] = clock
clock = clock + 1
```

Bài tập

Vẽ rừng DFS với cả số `pre` và `post` cho mỗi đỉnh cho đồ thị sau.



Tính chất của `previsit` và `postvisit`

Mệnh đề

Với mọi đỉnh u và v, hai khoảng

$$[\ \mathtt{pre}(u), \mathtt{post}(u)\ ] \text{ và } [\ \mathtt{pre}(v), \mathtt{post}(v)\ ]$$

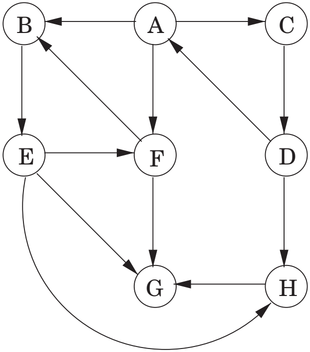
- hoặc là rời nhau,
- hoặc là có một khoảng chứa một khoảng khác.

Tại sao? vì $[\ \mathtt{pre}(u), \mathtt{post}(u)\ ]$ là khoảng thời gian đỉnh u nằm trong ngăn xếp. Cấu trúc vào-sau, ra-trước đảm bảo tính chất này.

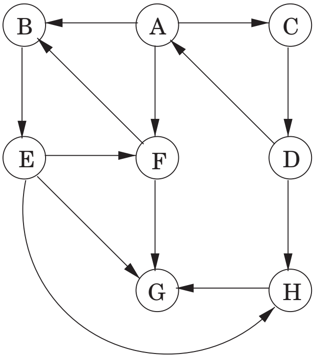
Nội dung

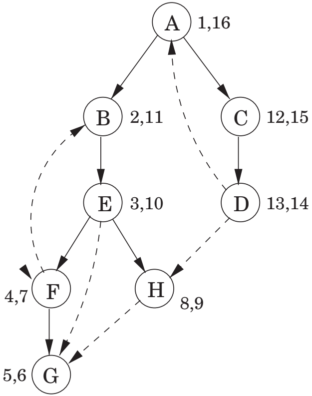
Bài tập

Hãy vẽ rừng DFS với số `pre` và `post` trên mỗi đỉnh cho đồ thị có hướng sau.

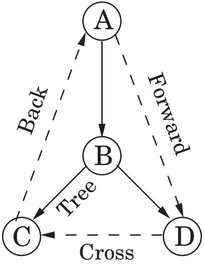


DFS trên đồ thị có hướng





Các kiểu cạnh



Tree Edges là cạnh thuộc rừng DFS.

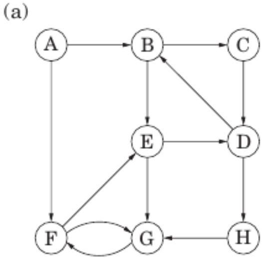
Forward Edges là cạnh dẫn từ một nút tới một nút con cháu của nó nhưng không thuộc rừng DFS.

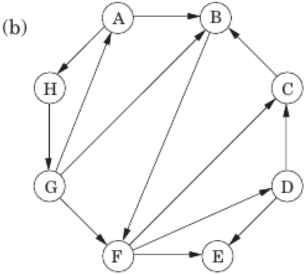
Back Edges là cạnh dẫn từ một nút tới một tổ tiên của nó.

Cross Edges là cạnh dẫn từ một nút tới một nút không phải tổ tiên cũng không phải con cháu của nó.

Bài tập

Thực hiện thuật toán DFS trên mỗi đồ thị sau; nếu phải thực hiện lựa chọn đỉnh, chọn đỉnh theo thứ tự từ điển. Phân loại mỗi cạnh (tree edge, forward edge, back edge, hay cross edge) và đưa ra số `pre` và `post` cho mỗi đỉnh.





Các khả năng cho cạnh (u, v)

Thứ tự `pre/post của` (u, v)				Kiểu cạnh
$\underset{u}{[}$	$\underset{v}{[}$	$\underset{v}{]}$	$\underset{u}{]}$	Tree/forward
$\underset{v}{[}$	$\underset{u}{[}$	$\underset{u}{]}$	$\underset{v}{]}$	Back
$\underset{v}{[}$	$\underset{v}{]}$	$\underset{u}{[}$	$\underset{u}{]}$	Cross

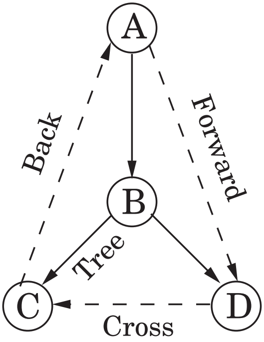
Câu hỏi

Tại sao các kiểu thứ tự khác không thể xảy ra?

Mệnh đề

Một đồ thị có hướng có chu trình nếu và chỉ nếu thuật toán tìm kiếm theo chiều sâu tạo ra back edge.

Chứng minh.



- Nếu (u, v) là back edge, thì $u \rightsquigarrow v \to u$ là một chu trình.
- Ngược lại, giả sử đồ thị có chu trình

$$C = v_0 \to v_1 \to \cdots \to v_k \to v_0.$$

Xét v_i là đỉnh đầu tiên trong C được thăm theo DFS. Mọi đỉnh khác trong chu trình sẽ đạt được từ v_i. Vậy thì $v_{i-1} \to v_i$ là back edge. $\square$

Bài tập

1. Hãy mô tả một thuật toán kiểm tra liệu đồ thị có hướng cho trước có chu trình hay không.
2. Hãy cài đặt thuật toán này.

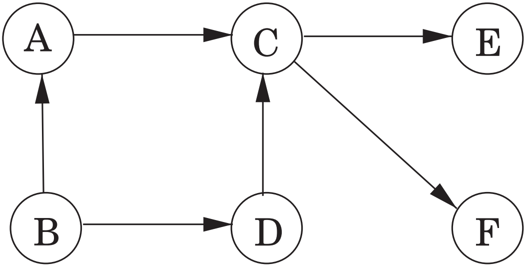
Đồ thị phi chu trình và sắp xếp topo

- Đồ thị phi chu trình (DAG) cho phép sắp thứ tự các đỉnh sao cho:

  Có cạnh (u, v) *nếu và chỉ nếu* u *đứng trước* v.

- Cách sắp thứ tự các đỉnh này gọi là *sắp xếp topo*.
- DAG cho phép mô hình hiệu quả các bài toán liên quan đến quan hệ nhân quả, phân cấp, phụ thuộc thời gian.
- Ví dụ: Mỗi môn học có môn tiên quyết (môn cần học trước). Một cách lựa chọn thứ tự học các môn là một cách sắp topo.

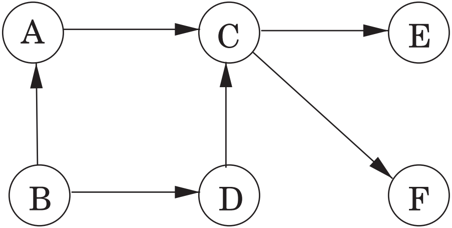
Đồ thị phi chu trình (DAG)



Phi chu trình $\iff$ *Không có Back Edge* $\iff$ *Sắp topo*

Bài tập

Hãy đưa ra mọi cách sắp topo cho đồ thị phi chu trình sau:



Tính chất của DAG

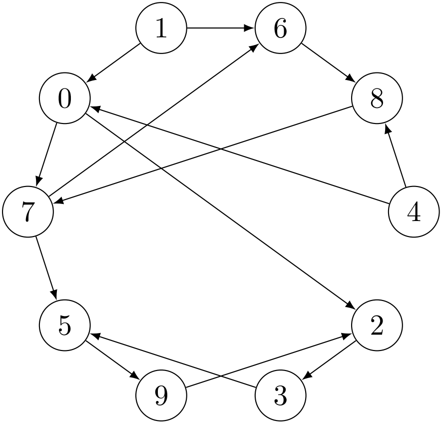
Mệnh đề

Trong DAG, nếu $(u, v) \in E$ thì $\texttt{post}(u) > \texttt{post}(v)$.

Vậy thì các đỉnh của DAG có thể sắp topo theo thứ tự giảm dần của `post`.

Bài tập

Xét một DAG có `pre` và `post` như dưới đây. Hãy đưa ra một thứ tự topo cho các đỉnh.

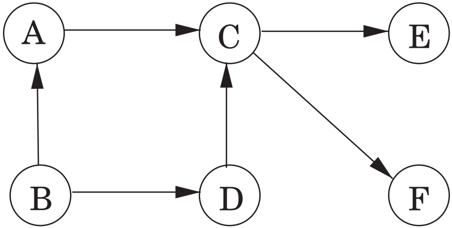


$0 : [1, 16]$
$1 : [17, 18]$
$2 : [2, 9]$
$3 : [3, 8]$
$4 : [19, 20]$
$5 : [4, 7]$
$6 : [11, 14]$
$7 : [10, 15]$
$8 : [12, 13]$
$9 : [5, 6]$

Bài tập

1. Hãy mô tả một thuật toán sắp Topo cho một DAG.
2. Hãy cài đặt thuật toán này.

Đỉnh nguồn và đỉnh hút



Trong đồ thị có hướng,

- Đỉnh nguồn (source) là đỉnh không có cạnh đi vào.
- Đỉnh hút (sink) là đỉnh không có cạnh đi ra.

Tính chất của DAG

Mệnh đề (Nhắc lại)

Trong DAG, nếu $(u, v) \in E$ thì $\texttt{post}(u) > \texttt{post}(v)$.

Vậy thì các đỉnh của DAG có thể sắp topo theo thứ tự giảm dần của `post`.

Và khi đó, đỉnh có `post` nhỏ nhất sẽ nằm cuối danh sách, và vậy thì nó phải là đỉnh hút.

Tương tự, đỉnh có `post` lớn nhất là đỉnh *nguồn*.

Mệnh đề

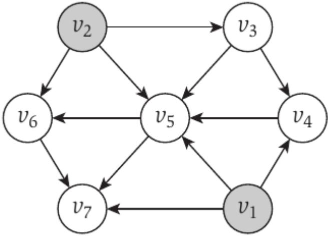
Mọi DAG đều có ít nhất một đỉnh nguồn và ít nhất một đỉnh hút.

Bài tập

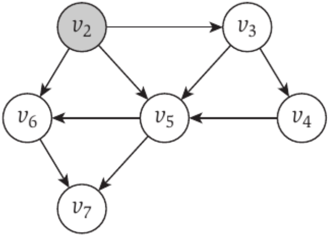
Hãy chứng minh mệnh đề trên.

Thuật toán sắp topo (thứ 2)

- Tìm một đỉnh nguồn, ghi ra nó, và xóa nó khỏi đồ thị.
- Lặp lại cho đến khi đồ thị trở thành rỗng.



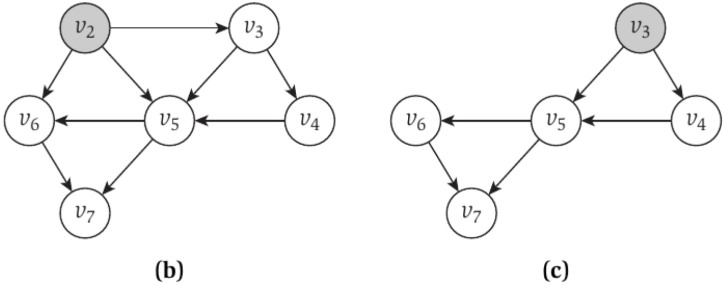
(a)



(b)

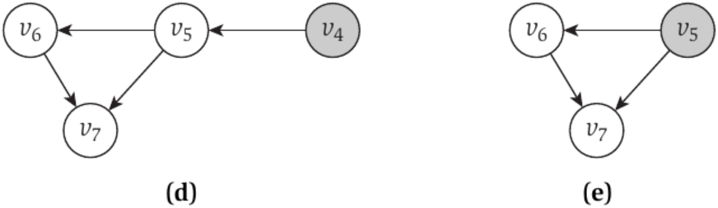
Thuật toán sắp topo (thứ 2)

- Tìm một đỉnh nguồn, ghi ra nó, và xóa nó khỏi đồ thị.
- Lặp lại cho đến khi đồ thị trở thành rỗng.



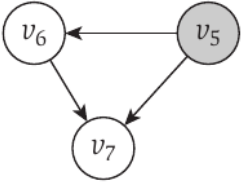
Thuật toán sắp topo (thứ 2)

- Tìm một đỉnh nguồn, ghi ra nó, và xóa nó khỏi đồ thị.
- Lặp lại cho đến khi đồ thị trở thành rỗng.

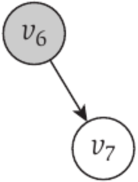


Thuật toán sắp topo (thứ 2)

- Tìm một đỉnh nguồn, ghi ra nó, và xóa nó khỏi đồ thị.
- Lặp lại cho đến khi đồ thị trở thành rỗng.



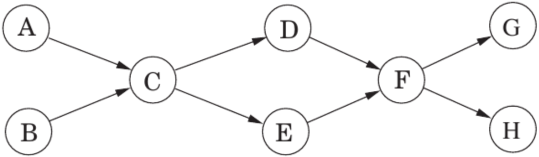
(e)



(f)

Bài tập

Chạy thuật toán sắp topo trên đồ thị sau:



1. Chỉ ra số `pre` và `post` của mỗi đỉnh.
2. Tìm các đỉnh nguồn và đỉnh hút của đồ thị.
3. Tìm thứ tự topo theo thuật toán.
4. Đồ thị này có bao nhiêu thứ tự topo?

Câu hỏi

- Tại sao thuật toán trước cho một thứ tự topo?
- Nếu đồ thị có chu trình thì thuật toán gặp vấn gì?
- Làm thế nào để cài đặt thuật toán này trong thời gian tuyến tính?

Nội dung

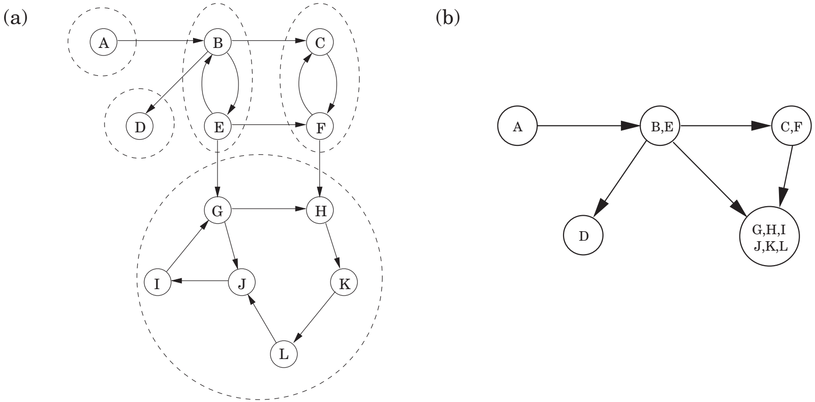
Thành phần liên thông mạnh

Định nghĩa

Hai đỉnh u và v của một đồ thị có hướng là ***liên thông*** nếu có một đường đi từ u tới v và một đường đi từ v tới u.

Quan hệ này phân hoạch tập đỉnh V thành các tập rời nhau và ta gọi các tập rời nhau này là các ***thành phần liên thông mạnh***.

Thành phần liên thông mạnh



Hình: (a) Đồ thị có hướng và các thành phần liên thông mạnh. (b) Các thành phần liên thông mạnh tạo thành một DAG

Các thành phần liên thông mạnh trong đồ thị

Mệnh đề

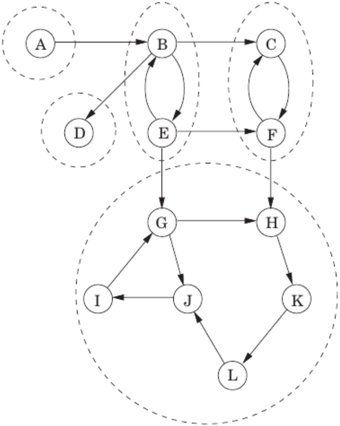
Mọi đồ thị có hướng đều là một DAG của các thành phần liên thông mạnh.

Vì nếu có chu trình đi qua một số thành phần liên thông mạnh thì các thành phần này phải được gộp chung lại thành một thành phần liên thông mạnh.

Một số tính chất

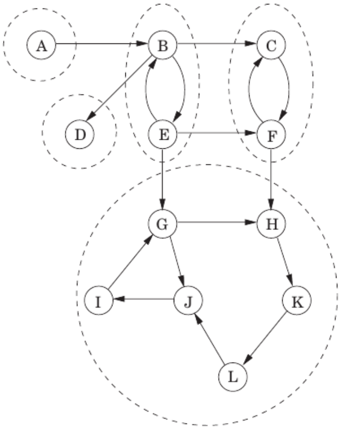
Mệnh đề

Nếu thủ tục con `explore` bắt đầu từ một đỉnh u, thì nó sẽ kết thúc khi mọi đỉnh có thể đến được từ u đã được thăm.



Một số tính chất

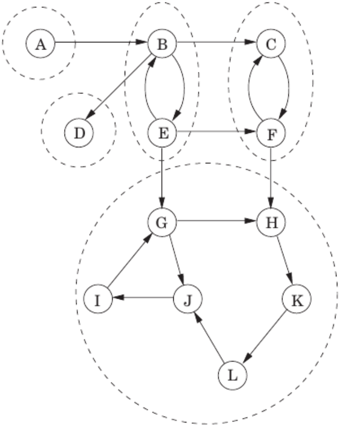
Nếu ta gọi `explore` từ một đỉnh thuộc một thành phần liên thông mạnh hút, vậy thì ta sẽ nhận được đúng thành phần này.



Câu hỏi

1. Làm thế nào ta có thể tìm được một đỉnh mà ta chắc chắn nó thuộc vào thành phần liên thông mạnh hút?

2. Ta sẽ tiếp tục thế nào khi đã tìm được một thành phần liên thông mạnh?



Câu hỏi 1

Làm thế nào ta có thể tìm được một đỉnh mà ta chắc chắn nó thuộc vào thành phần liên thông mạnh hút?

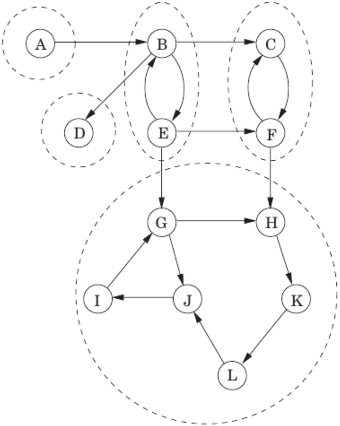
Xét G^R là đồ thị ngược của G, tức là đồ thị G^R đạt được từ G bằng cách đảo hướng các cạnh.

Thành phần liên thông mạnh của G cũng là của G^R. Tại sao?

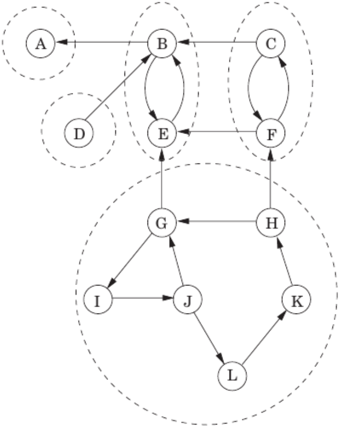
Và thành phần liên thông mạnh hút trong G sẽ là thành phần liên thông mạnh nguồn trong G^R.

Mệnh đề

Đỉnh có số post lớn nhất theo DFS phải thuộc một thành phần liên thông mạnh nguồn.



Hình: Đồ thị G



Hình: Đồ thị ngược G^R của G

Câu hỏi 2

Ta sẽ tiếp tục thế nào khi đã tìm được một thành phần liên thông mạnh?

Mệnh đề

Nếu C và D là các thành phần liên thông mạnh, và có một cạnh từ một đỉnh trong C tới một đỉnh trong D, vậy thì số `post` lớn nhất trong C phải lớn hơn số `post` lớn nhất trong D.

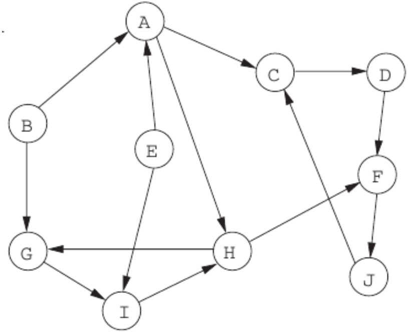
Khi ta tìm thấy một thành phần liên thông mạnh và xóa nó khỏi đồ thị G, vậy thì đỉnh với số post lớn nhất trong các đỉnh còn lại sẽ thuộc vào thành phần liên thông mạnh hút của đồ thị còn lại của G.

Thuật toán tìm thành phần liên thông mạnh

1. Chạy DFS trên đồ thị ngược G^R của G.
2. Chạy thuật toán tìm thành phần liên thông (tương tự như của đồ thị vô hướng) trên đồ thị có hướng G; và trong khi chạy DFS, xử lý các đỉnh theo thức tự giảm dần theo số `post` của mỗi đỉnh.

Bài tập

Chạy thuật toán tìm thành phần liên thông mạnh trên đồ thị sau (theo thứ tự từ điển).



Bài tập

Chạy thuật toán tìm thành phần liên thông mạnh trên đồ thị sau (theo thứ tự từ điển).